Lazy Updating of hubs can enable more realistic models by speeding up stochastic simulations

Kurt Ehlert[1,2] and Laurence Loewe[1,2,a)]
[1]*Laboratory of Genetics, University of Wisconsin-Madison, Madison, Wisconsin 53706, USA*
[2]*Wisconsin Institute for Discovery, University of Wisconsin-Madison, Madison, Wisconsin 53715, USA*

(Received 8 October 2013; accepted 24 October 2014; published online 26 November 2014)

To respect the nature of discrete parts in a system, stochastic simulation algorithms (SSAs) must update for each action (i) all part counts and (ii) each action's probability of occurring next and its timing. This makes it expensive to simulate biological networks with well-connected "hubs" such as ATP that affect many actions. Temperature and volume also affect many actions and may be changed significantly in small steps by the network itself during fever and cell growth, respectively. Such trends matter for evolutionary questions, as cell volume determines doubling times and fever may affect survival, both key traits for biological evolution. Yet simulations often ignore such trends and assume constant environments to avoid many costly probability updates. Such computational convenience precludes analyses of important aspects of evolution. Here we present "Lazy Updating," an add-on for SSAs designed to reduce the cost of simulating hubs. When a hub changes, Lazy Updating postpones all probability updates for reactions depending on this hub, until a threshold is crossed. Speedup is substantial if most computing time is spent on such updates. We implemented Lazy Updating for the Sorting Direct Method and it is easily integrated into other SSAs such as Gillespie's Direct Method or the Next Reaction Method. Testing on several toy models and a cellular metabolism model showed >10× faster simulations for its use-cases—with a small loss of accuracy. Thus we see Lazy Updating as a valuable tool for some special but important simulation problems that are difficult to address efficiently otherwise.
[http://dx.doi.org/10.1063/1.4901114]

I. INTRODUCTION

The behavior of complex dynamic biological interaction networks is often difficult to understand. Time series can help by showing how the amounts of important entities in a network change over time. They are used in many areas of biology, including biochemistry,[1–5] ecology, evolution,[6,7] and systems biology.[1,2] Since simulation techniques for predicting time series are of general interest, we will discuss models in terms of "parts" and "actions" instead of "molecules" and "reactions." Depending on the context, "parts" can stand for molecules in a cell, individuals in a population, or organisms or nutrients in an ecosystem. Likewise, "actions" can be as diverse as biochemical reactions in a cell or predator-prey interactions in the wild, as long as they map to the underlying mathematical representation. Time series are often computed with a deterministic ordinary differential equation (ODE) solver or a stochastic simulation algorithm (SSA). Which option is better depends on model details.

If all parts in a system are always present in very large numbers, ODEs produce time series of good accuracy at high speed, using well established algorithms[8,9] that have been applied to diverse areas of biology.[1,2,6] Deterministic ODE models work well at such large part amounts, as stochastic effects become relatively less important. However, many processes in biology are discrete and involve very few key parts, such as switching on single-copy genes or single viral particles that cause dramatic growth but only if they infect a cell. Stochastic effects are important in these systems[3–5] and simulation models should respect the discreteness of molecules, individuals and unique events.

Continuous-time Markov chains[10–15] (CTMCs) have a track record of excellence in this area and provide a rigorous basis for transforming certain types of well-mixed stochastic models with infinitely large part numbers into equivalent ODE models.[3,14,15] CTMCs have no "memory," as they assume that the future state of a model only depends on its current state and not on its history. CTMCs also assume that the waiting times between actions (which lead to state transitions) are exponentially distributed, unless external noise affects the probabilities that determine how often these actions occur. In that case, waiting times are no longer necessarily exponentially distributed, as they are modulated by current external changes and the Markov process becomes time inhomogeneous.[16,17] In biochemical systems such external noise can be caused by fluctuations in volume or temperature, assuming both are not affected by reactions in the system (see below). Such external noise can often be approximated well by a CTMC jump process if the time between actions is small compared to the jumps (see Refs. 17–19 and pp. 327–332 in Ref. 17).

a)Author to whom correspondence should be addressed. Electronic mail: loewe@wisc.edu

SSAs compute individual stochastic realizations of time series that are guaranteed to exactly reflect the probabilities in CTMCs.[3,10–12,17,20–22] For each action that occurs, typical SSAs compute at least one randomly distributed waiting time, which follows an exponential distribution if the CTMC is time homogeneous. Thus expected waiting times are determined by a single parameter, the "propensity," which is proportional to the probability that an action will occur within the next infinitesimally small time step.[10] The correctness of SSAs depends on the precision of propensity calculations. Exact SSAs recompute one or more propensities almost every time an action occurs[3,10–12,17,20–22] and thus need much more computing time than ODEs for systems with many parts. Computing time problems for SSAs are exacerbated by the stochasticity of simulation results, as many runs are required for inferring summary statistics. As a result, much effort has been invested in the speed-up of exact or approximate SSAs,[18,23–36] producing different algorithms that require different tradeoffs.[18,24–40] Tau-Leaping has emerged as an important approximate SSA.[21,24,37,41–43] Below we present an algorithm that trades a very small amount of accuracy for a substantial reduction in computing time by skipping some updates in a special category of parts, which we call "hubs." The usefulness of such an approximation depends on the biological importance of hubs. Next we explain what hubs are and why they are important for a substantial number of important biological modeling questions.

Hubs. If a part's amount affects the propensities of many actions, then we recommend referring to that part as "highly connected," "hub part" or simply as "hub." Such hubs are common in biology and include well-known molecules like water or ATP (a key cellular energy hub), both of which affect numerous reactions.[44] Biochemists know many more hub molecules, including co-enzymes, H^+ ions that impact many cellular processes by determining pH,[2] and hub proteins.[45] The easiest strategy for integrating hubs into models is to assume that their amount does not change and contributes to a constant environment that incorporates them implicitly into the rate coefficients of the model. We can increase model flexibility by adding hubs explicitly to models as *externally controlled* quantities that affect reactions; this is computationally efficient but only realistic if external control mechanisms are appropriate. However, if many reactions in a model can affect hubs in complicated ways, then hubs need to be fully integrated into the model, lest we risk missing critical roles of hubs that can affect important high-level outcomes like cell growth. For example, if the amount of dNTPs was assumed to be constant in a recent whole-cell simulation of *Mycoplasma genitalium*, then it would have been impossible to find that dNTP synthesis was limiting DNA synthesis and thus cell growth.[46]

Volume. Hubs do not necessarily have to be molecules. For example, reaction volume affects many propensities.[3,16,21] If the volume is affected in small steps by many actions in the system, then its impact on propensities is similar to that of hubs: frequent small changes in volume trigger frequent small updates to many propensities. Biological examples for such endogenously driven changes include active cellular nutrient import,[44] osmosis,[44] and export of molecules.[44] They all change cell volume frequently, but only by minuscule amounts. Over time these small effects combine to produce visible phenomena such as the growth of cells until doubling[47,48] or the shrinking of cell size in excessively salty solutions due to osmosis.[49–51] Proposed mechanisms for the interaction between cell volume and the molecular control of the cell cycle that triggers the production of daughter cells include sensing a change in cell size by sensing changes in some molecular concentrations.[52] These aspects of cell biology are sometimes integrated into simulation models;[18,19,23] considering them more often would be facilitated by more efficient ways of simulating hubs and could help us understand a diverse array of evolutionary questions including how cancer cells grow.

Temperature is also well known to affect reaction rates and the strategies that might motivate treating volume as a hub are easily adapted to temperature. But does temperature matter as a hub or is it mostly controlled externally? Temperatures for the biochemistry in cells are determined by complex interactions between (i) external environmental temperatures, (ii) the physics and geometry of heat transfer, (iii) the temperature regulating circuits and mechanisms an organism might have and (iv) the heat generated by the metabolism of the cell.[53] While assuming a constant or externally controlled temperature is often reasonable, a number of recent results have shown NAD+ dependent deacetylase Sirtuin-1 to be a conserved key player in controlling overall metabolic activity in organisms from yeast to humans.[54,55] Thus, a protein in the cell regulates how many biochemical reactions per second contribute to heating the cell, which in turn affects reaction rates. Modeling such temperature control might have to track many minuscule amounts of heat. How big are errors in biochemical rates, if we ignore temperature? A simple calculation might provide a useful intuition using observed Q_{10} temperature coefficients (predict fractional rate change if temperature is increased by 10 °C; see Arrhenius equation). Assuming $Q_{10} \approx$ 2-3 as typical[56,57] we can assess the increase in human metabolic rates expected from assumed temperature differences caused by normal circadian fluctuations (36 + 2 °C, see Ref. 58) or from strong fever (36 + 5 °C) to be 15%–25% or 41%–73%, respectively. Notwithstanding higher-level regulatory circuits, these commonly observed increases in body temperature are ultimately generated by cells that burn more energy. The metabolism of such cells is profoundly affected by both, regulatory molecules such as Sirtuin-1 and increases in temperature from its own metabolic activity. Other systems where metabolism and heat are intrinsically linked include compost heaps[59] and industrial plants used for bioenergy production,[60,61] where microbes break down organic matter and generate substantial heat that speeds up reactions that generate more heat (until heat degrades enzyme activity or diffuses away). Investigating temperature dependencies in such systems using standard SSAs is computationally costly due to the hub-like properties of temperature.

Biological evolution critically depends on fitness correlates such as the growth and survival of organisms. Population genetic analyses have shown that selection in large populations over many generations is very successful at selecting very small differences in fitness—often too small for

direct measurements in the lab.[6,62] A key goal of evolutionary systems biology analyses is to enable the computational prediction of candidate fitness correlates based on the best systems biology models available by integrating all relevant data using the most rigorous analyses available.[63,64] For this goal, it is crucial to integrate hubs such as ATP, volume, and temperature into systems biology models, because the rate of cell growth and the energy spent on keeping a cell alive are important factors that can affect the growth and survival of organisms.[64] Mutations that affect hubs and thereby cell growth or survival by as little as 0.1% can already be major drivers of evolution, as many important outcomes in evolution depend on a fine balance between many small actions that slowly increase or decrease a quantity on the longer-term (e.g., no growth of cells without slowly changing cell volume; no survival in some climates without adjusting body-heat by increasing metabolic rates). Thus the routine inclusion of known hubs into dynamic systems biology models will help prepare such models for computing fitness correlates at higher precision.

For the reasons above it is desirable to have a method in the growing toolbox of stochastic simulation algorithms that is particularly well suited for addressing hubs. Without such a method, promising lines of enquiry may be computationally prohibitive.

Here we present "Lazy Updating," which is an add-on for many existing SSAs. It reduces computing costs caused by the frequent propensity updates that can be triggered by hubs. Lazy Updating skips all propensity updates triggered by hubs until the amount of the hub changes beyond a specified threshold. We tested this on the following models: (A) an artificial ATP pathway, with ATP as hub to highlight strengths of Lazy Updating. (B) Birth and (C) Consumption models contain only one action to test the accuracy of Lazy Updating at known weak points. (D) Detailed prototypic models of metabolism are used to test how Lazy Updating performs "in the wild." We expected and found (A) and (D) to work well with Lazy Updating; accuracy was easy to control and even models it was not designed for showed small speedups. Overall, we find hardly any costs to Lazy Updating, but huge wins if frequently updated hubs are involved, making Lazy Updating a useful addition to the growing toolbox of stochastic simulation methods.

II. METHODS

A. Dependency structure in the computation of propensities

System overview: We consider a CTMC describing a system of parts whose amounts are affected by actions. The amounts of the corresponding "required parts" are used to calculate each action's propensity, which is proportional to the probability that this action will occur within the next infinitesimally small time step. When an action occurs, it affects the state of the system by changing the amounts of some parts, which we call "changed parts," an action-specific set consisting of "consumed parts" and "produced parts." Examples include substrates and products of biochemical reactions, as well as other quantities such as volume or temperature that effectively behave like parts when they are changed by actions. If the "changed parts" of an "executed action" A_1 are "required parts" for another action A_2, then A_2 is an "affected action" of A_1, because the propensity of A_2 is affected by executing A_1. As described below, Lazy Updating is reasonable if changes are small enough compared to existing amounts of the corresponding parts.

Notation: We consider a CTMC model $\boldsymbol{M}$ containing ($\boldsymbol{P}$, $\boldsymbol{p}$, $\boldsymbol{A}$, $\boldsymbol{a}$, $\boldsymbol{pr}$, $\boldsymbol{pc}$, $\boldsymbol{r}$, t), where t is the simulated time measured in units shared among all time-related values of the system, which is defined as follows. Let n represent the number of parts, and let P_i denote the ith part. The amount of P_i is denoted as p_i. The set of all parts in the system is $\boldsymbol{P}$, the set of all corresponding amounts is $\boldsymbol{p}$. Likewise, the kth of m actions is A_k and it occurs with propensity a_k, where $\boldsymbol{A}$ and $\boldsymbol{a}$ denote the corresponding sets of all actions and their propensities in the system, respectively. The propensity a_k is a function that is evaluated at a given time t to calculate a value proportional to the probability that A_k will occur in the next infinitesimally small moment in time. It depends only on the "required parts" of *this* action and on its rate coefficient r_k (or other values in a_k if A_k does not follow mass action dynamics or depends on external noise). Let $p_{r(i,k)}$ represent the required discrete individual units of parts of type P_i for a single occurrence of action A_k, and let $p_{c(i,k)}$ be the change in the amount of part P_i that is caused by that single occurrence. Both are zero for parts that are not involved in this action. However, while $p_{c(i,k)}$ can be negative (consumed) or positive (produced), $p_{r(i,k)}$ cannot be negative. Both may be combined into a stoichiometry matrix (see p. 28 in Ref. 16).

Propensities: Generally, the propensities, $\boldsymbol{a}$, of all actions $\boldsymbol{A}$, are functions of the state of the system including potential external noise during the time interval from after the last and before the next action. For this duration the state of the system is characterized by the constant amounts, $\boldsymbol{p}$, of all parts, $\boldsymbol{P}$, in the system if we assume no external noise. These definitions have been at the heart of CTMC and SSA theory for a long time.[10–12,15,16,20,21]

After an action occurs, the amounts of some parts change, forcing a recalculation of the propensities in the system. These recalculations are often very expensive and take up a large fraction of a simulation's computing time. Lazy Updating aims to reduce this cost. To do so, it is helpful to understand more about the internal dependencies between propensities and amounts of parts:[18]

1. Not *all* propensities in $\boldsymbol{a}$ depend on the amounts of *all* parts in $\boldsymbol{p}$; rather each propensity only depends on the amounts of required parts; for these parts we have $p_{r(i,k)} > 0$. For most actions the size of the list of required parts is very small compared to n.
2. Not all propensities in $\boldsymbol{a}$ need updating after the next action A_k occurred. Rather, we need to update only the propensities of 'affected actions' that depend on parts that were changed by A_k; for these parts we have $p_{c(i,k)} \neq 0$.
3. For most actions the list of affected parts is very short.
4. Most parts are only required by few actions, so updating the corresponding propensities does not seem very costly.

5. However, some parts are required by many actions. Updating all affected propensities after a single change in the amount of these parts can trigger many computations. If these parts are also affected by frequently occurring actions, then a large fraction of computing time could be consumed by tracking the propensities of these parts. These are what we refer to as "hub parts" or "hubs."

Now we will take a closer look at the standard function for computing the propensity in mass action models as used here (see also p. 182 in Ref. 16):

$$a_k = r_k \prod_i \frac{p_i!}{(p_i - p_{r(i,k)})! p_{r(i,k)}!}, \tag{1}$$

where r_k is scaled appropriately using a "volume correction" given in Refs. 12 and 16. The term inside the product is the number of combinations of $p_{r(i,k)}$ distinct parts that can be chosen out of the total amount p_i (following p. 181 in Ref. 16). For parts in $\boldsymbol{P}$ that are not required parts and hence do not contribute to a_k (i.e., $p_{r(i,k)} = 0$), the corresponding term in the product evaluates to 1; thus we can skip over them when implementing the algorithm, and whether we compute Eq. (1) using all parts or using only the required parts is not a matter of correctness, but one of speed. If an action A_k requires only one single unit out of p_i such units of a part, then $p_{r(i,k)} = 1$ and the term evaluates exactly to p_i.

Temperature and volume as parts of the system: Below we discuss the detailed Model D, where we use the same "volume correction" as above[12,16] to make r_k a function that tracks changes in volume over time. This effectively turns the volume into a "non-standard part" in $\boldsymbol{M}$, because we cannot use Eq. (1) to compute a single factor by which the volume scales propensities of actions of different order. Instead, we must use special functions (here volume correction[12,16]). Also, the volume amount is usually not measured in units of 1 as counts of molecules would be. Temperature is another important factor affecting the speed of all chemical reactions;[2] therefore we explore a relative temperature correction for r_k, where the relative temperature, T_r, has an initial relative amount of 1. T_r in $\boldsymbol{M}$ is also likely to assume non-integer values, like the volume. To scale propensities with T_r we simply multiply the result of Eq. (1) for all standard parts by T_r (which is equivalent to including T_r as a standard part into Eq. (1) assuming $p_{r(i,k)} = 1$). We note that a transformation between T_r and measurable temperatures may be required for building more realistic models.

Changes of volume and T_r are defined as for all other parts by choosing a corresponding $p_{c(i,k)}$ depending on A_k and P_i, where P_i stands for volume or T_r. In our study all changes of volume and T_r are in discrete steps caused by discrete actions; hence the overall dynamics of changes in $\boldsymbol{M}$ is still exclusively defined by propensities that only depend on the amounts of parts. Thus nothing except time changes while $\boldsymbol{M}$ waits for the next action (in absence of external noise). These dependencies turn neither volume nor T_r into hubs. However, if every molecular import/export reaction of a cell changes cell volume and every chemical reaction releases or consumes very small amounts of energy,[2] then volume and T_r are changed by many reactions (each of which triggers propensity updates for many affected reactions). In these cases, both volume and temperature are likely to be hubs as defined here.

B. The Lazy Update algorithm

The structure of propensities as captured in Eq. (1) provides an opportunity for separation of concerns, namely between:

1. Amounts of parts that usually change by a substantial relative amount, resulting in a substantial change in propensity for all affected actions that depend on these parts and thus require an immediate update of all these propensities.
2. Amounts of parts that usually change by a very small relative amount, resulting in a minuscule change in propensity for all actions that depend on these parts; these open the possibility for accumulating many small changes before triggering costly propensity updates. Skipping minuscule propensity updates and only performing an update when it really matters is the essence of Lazy Updating.

A naïve SSA implementation recalculates *all* propensities whenever the state of the system changes. More advanced implementations only recalculate the propensities that are actually affected by the parts that just changed with the execution of the last action,[18,26] which speeds up simulations substantially, but does not help calculate propensities of highly connected parts whose amounts change frequently, where each change triggers numerous propensity updates. Lazy Updating reduces the burden from these propensity updates.

If an SSA updates all necessary propensities *every* time *any* reactant amount changes, then we say that the SSA uses "**Immediate Updating.**" In contrast, we denote as "**Lazy Updating**" any algorithm that postpones some propensity updates until the amount of a part has changed enough to cross a defined threshold. In principle, such a threshold can be defined in many different ways; here we choose to fix it at some percentage of the old amount of a part that was stored when the last complete propensity update was triggered. This facilitates a useful intuition explained below and requires simulations to specify the **relative Lazy Updating error tolerance** ε_{Li} for part i (or ε_L for all parts that are updated lazily in $\boldsymbol{M}$). It defines a condition that forces propensities to update and specifies a range of acceptable errors in amounts that affect propensities from missing updates.

Let $p_i(t\text{-}\Delta t)$ be the amount of part i at the time $t\text{-}\Delta t$ when the last propensity update was triggered by this part. Then an update of all the part's dependent propensities is triggered when all combined changes exceed the threshold defined by

$$|p_i(t - \Delta t) - p_i(t)| > \varepsilon_{Li} p_i(t - \Delta t). \tag{2}$$

Thus smaller ε_{Li} result in more frequent propensity updates. If the tolerance is set to a relative error of $\varepsilon_L = 0.1\%$, then Lazy Updating automatically reverts to exact Immediate Updating whenever the amount of a discrete part is below 1000 individual particles, as every change in amount below that

```
// Pseudocode SSA: Immediate Updating (IU) with Lazy Updating (LU)
Set initial conditions for model: amounts, propensities, times,...

// Map from each Action to all Actions it affects and that require IU
// These dependencies are updated independently from Parts
Initialize for each Action: Links to each Action_Requiring_IU

// Map from each LU Part to all Actions whose propensities it affects
Initialize for each Part_Allowing_LU: Links to each Action_Allowing_LU

While(Time_Current < Time_End) {
   Choose Time_To_Next_Action
   Time_Current += Time_To_Next_Action
   Choose next Action A
   Execute A: Apply all its AmountChanges to all its ChangedParts
   Update Propensities of each Action_Requiring_IU in A
   For each ChangedPart P whose Amount is changed by Action A {
      If( P is a Part_Allowing_LU ) {
         P.ChangeSinceUpdate = P.AmountNow - P.Amount_At_Last_Prop_Update
         If( |P.ChangeSinceUpdate| > P.Amount_At_Last_Prop_Update * P.εL){
            Update Propensities for each Action_Allowing_LU in P
            P.Amount_At_Last_Prop_Update = P.AmountNow
         } // EndIf:  Propensity Updates were not needed
      }    // EndIf:  Lazy Updating was not enabled
   }       // Next ChangedPart Goto 16
}          // Next Time           Goto 10
```

FIG. 1. **Pseudocode of a simple Lazy Updating algorithm.** Bold lines denote the Lazy Update additions embedded in an example SSA. The main idea is to tolerate small inaccuracies in propensities caused by minute changes in the amounts of parts, until inaccuracies cross a threshold triggering updates of all relevant propensities.

threshold triggers a propensity update (a similar intuition for non-discrete parts such as volume and temperature is more complicated). If the error tolerance is zero, then a Lazy Updating algorithm effectively switches itself off.[79] Looking at the tolerance from that perspective provides a useful intuition for choosing appropriate values in a number of scenarios.

A simple algorithm for Lazy Updating is presented in Figure 1. Making Lazy Updating efficient requires data structures that quickly assess which actions are affected by a change in a certain part. Other than that, requirements are minimal, so Lazy Updating fits well into the general structure of many SSAs. It can therefore work with a broad range of relevant algorithms, including Gillespie's Direct Method,[11] the Next Reaction Method,[18,23] and others. We implemented Lazy Updating within the Sorting Direct Method,[27] which is a variant of Gillespie's Direct Method.[11]

C. Models used for testing Lazy Updating

To measure the accuracy and speed of Lazy Updating, we applied it to three types of simple models that test Lazy Updating in specific ways (Fig. 2; between 1 and 1002 actions), and three more detailed prototypes of a metabolic model to represent the higher complexity from more realistic models (Fig. 11; 92 parts, 188 actions). Each of these models is represented by a different CTMC system $\boldsymbol{M}$ as defined above and was implemented for simulation in an early prototype of the Evolvix model description language and simulation infrastructure (http://evolvix.org) that was modified to support Lazy Updating. Next we describe the various models and the aspects of Lazy Updating they were designed to test, along with the parameters we used in our simulations.

Model A: The ATP Model (Fig. 2(a)) consists of two different versions, A_1 and A_2, where A_1 contains a fast reversible reaction that involves a single transcription factor molecule and a single DNA molecule that are not linked to the ATP pathway; A_2 does not have the reversible reaction. The purpose of that reaction is to model situations where many other actions in $\boldsymbol{M}$ are not hub-dependent *and* frequently occurring *and* involve rare parts (all three conditions satisfied). Here we are not interested in the individual identity and effects of these actions; thus we lump them all together into one single action that occurs at a correspondingly higher rate. Examples for individual actions might include transcription factors that bind to a specific DNA binding site that exists only once or twice in a cell. We approximate the combined effect of all these actions on the speed of Lazy Updating or Tau-Leaping[24,42] by including just that one independent action in A_1. This model is not meant to be realistic in detail, but rather to reflect a range of potentially realistic hub

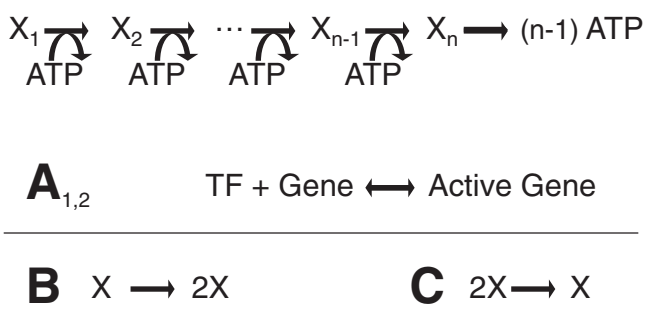

FIG. 2. **Illustrative toy models used.** (a) Two **ATP Models** that both loosely capture ATP's role as ubiquitous cellular energy hub by including many reactions that depend on ATP and that all require propensity updates whenever one of them occurs. To investigate the potential impact of other reactions, we added a fast reversible binding reaction between a transcription factor and a single gene in model A_1, both of which are missing in model A_2. (b) The **Birth Model** was chosen to find weak spots in Lazy Updating. We expect and see delayed growth from consistent slight underestimates of propensity by Lazy Updating as parts can only increase. (c) The **Contraction Model** is the reverse of the Birth Model. Decreasing amounts cause Lazy Updating to overestimate propensity and slightly delay contraction; errors become irrelevant when approaching zero. See text for parameters of all models and more details.

sizes in two scenarios of different propensity updating patterns. This was designed to help identify when Lazy Updating might lead to larger speedups than other simulation methods such as Tau-Leaping, which is slowed down by rare parts involved in frequent actions.[43] We used this model to compare the Sorting Direct Method with Lazy Updating to the Partial Propensity method SPDM, Tau-Leaping, the Optimized Direct Method, and our own Sorting Direct Method with Immediate Updating (see below for details). We varied the number of reactions in the ATP pathway to see how the different algorithms scaled with the number of actions and propensity updates.

In the model, the initial amount of ATP is 10^6, and the initial amount of each molecule X_i is 100. The rate coefficients for the reactions in the pathway component are all 10^{-5} per time unit, except for the last reaction, which has a rate coefficient of 10. For the fast reversible binding reaction, the forward and reverse rate coefficients are both 10 000. We chose the rate coefficients such that the pathway starts at its steady-state, and we set the volume to 1. For every test, the model was run until a simulation time of 10 arbitrary time units. We measured the amount of X_7.

Model B: The Birth Model only has a single action that replicates part X (Fig. 2(b)). It was chosen to highlight potential problems with the accuracy of Lazy Updating. Since the parts are always produced and never consumed, errors never cancel out and the propensity is always underestimated. We therefore expected Lazy Updating to accumulate large accuracy errors for the Birth Model, making it an excellent test case highlighting important limits for the accuracy of results generated by Lazy Updating. The rate coefficient was 0.4 per time unit, the volume was 1, and the initial amount was 10. The simulations were run until 30 time units.

Model C. The Contraction Model (Figure 2(c)) is the reverse of the Birth Model. The part is only consumed resulting in a contraction of the population, so Lazy Updating consistently overestimates the propensity of the reaction. As with the Birth Model, we choose the Contraction Model to test the accuracy of Lazy Updating. The rate coefficient was 0.001 per time unit, the volume was 1, and the initial amount was 10 000. The simulations were run until 1 time unit.

Model D. This **Detailed Metabolic Model Prototype** is a real-world test case for Lazy Updating. Fig. 11 shows how its 92 parts connect to effectively 188 actions (in reversible pairs) in a network structure more complex than for our simple models. This prototypic metabolic model was constructed by using techniques that combined metabolomics measurements of small molecule concentrations with stoichiometric reaction networks in an attempt to reconstruct parts of the metabolism of human red blood cells.[65] Formally, this resulted in a mass action model that incorporates kinetics and regulation into what otherwise would have been a purely stoichiometric model as used in Flux Balance Analysis.[66] The final model has 94 named reversible mass action reactions with rate coefficients and deterministic ODE equilibrium concentrations of the 92 named metabolites. We extracted the data (from the supplementary material of Ref. 65) and converted it into three Evolvix model description files for stochastic simulations, which addressed the following three different modeling scenarios: (i) How much speedup does Lazy Updating provide for known hubs in the known network? (ii) Is this changed if all actions that import or export molecules change the volume and ∼50% of all actions are non-first-order actions that depend on the volume? (iii) What if temperature is increased whenever an action occurs and all actions depend on temperature?

We set the effective volume of the system to 10^6 for all our simulations such that the number of glucose molecules in our model was scaled to about 5×10^6 per cell (which approaches realistic numbers). We dynamically applied standard approaches for correcting reaction rates to account for the volume[12,16] whenever we computed propensities. The volume increased by an assumed amount of 10^{-5} whenever a molecule was imported. We chose our total simulated model time (0.1 ms) such that it allowed for effective speed measurements, but did not allow volume or temperature to substantially change the system away from the equilibrium state that it originally had. To model temperature, we assumed that each action increased the relative temperature T_r by the small amount of 10^{-9} (as discussed above). This is to reflect the fact that chemical reactions are either exothermic or endothermic, and thus can in principle affect temperature. Temperatures in our simulations start with a value of $T_r = 1$ that indicated no effective temperature change relative to the original model. All propensities are simply multiplied with this effective relative temperature and are thus affected as temperatures change. While we recognize the lack of thermodynamic realism in this version of the model and its changes to T_r, it does capture a dependency structure of interest for processes that affect the volume or contribute to local heating in biological tissues (see Sec. I). Formally, this scheme for modeling volume and temperature as parts can be described by $\boldsymbol{M}$ as defined above.

We sorted the metabolites of the model into 3 classes as indicated by color coding (Fig. 11, zoom in for details): "5Hubs" are parts required in at least 5 reversible

reactions; "1KFast" parts have a predicted equilibrium concentration that scales to an amount of 1000 or more; all remaining metabolites were designated "normal." We applied Lazy Updating for different combinations of those distinct classes of molecules, and defined three different scenarios to mimic ways how hubs might appear in modeling studies: **Network-connectivity** is already defined by the structure of the network and is explored by activating Lazy Updating for 5Hubs only. **Volume-connectivity** added minuscule changes in volume affecting the propensity of about half of all actions, as caused by all import reactions. **Temperature-connectivity** included the relative temperature T_r as defined above in all actions in addition to the volume changes. While these amounts are often too small to warrant further investigation, their combined effects could be substantial. This to link microscopic energy budgets with gene regulatory networks and macroscopic measurements in simulations as facilitated by Lazy Updating could open new lines of inquiry (see Sec. I).

D. Estimating the Speedup from Lazy Updating

For each model, we compared the speed of Immediate Updating to the speed of Lazy Updating with a range of Lazy Updating tolerances ε_L. The run time was measured with Python's *time* function, and each algorithm was instructed to record as little time series data as allowed by the implementations to allow measuring speeds of algorithms without complications from data storage. For Models A, B, C, and D, we averaged run time measurements over 50, 5000, 5000 and 3 simulations, respectively. For Model A, we tested $\varepsilon_L = \{0.25, 0.5, 1, 2, 3, 4, 5, 7.5, 10, \text{and } 12.5, \text{all scaled} \times 10^{-5}\}$, for Model B, $\varepsilon_L = \{10^{-4}, 3\times10^{-4}, 10^{-3}, \ldots, 0.1, 0.3, 1, 2, 3, \ldots, 9, 10, 30 \text{ all scaled} \times 10^{-2}\}$, for Model C, $\varepsilon_L = \{0.5, 1, 2, 3, \ldots, 7, 8, \text{all scaled} \times 10^{-3}\}$, and for Model D, $\varepsilon_L = \{10^{-9}, 10^{-8}, 10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}\}$.

Ideally, we would like to predict the speedup based on ε_L. However, it is unlikely that this will be possible except for artificially simple models, as any real speedup is deeply intertwined with the structure of a model and the parameter combinations used to run it. Nevertheless, we can use four observed run times to construct an equation that approximates the speedup in a useful way for some simple models. With the parameterized equation in hand, it becomes easier to navigate the tradeoffs between speedup and accuracy, as optimal tradeoffs depend on the questions the model was constructed for. The Lazy Updating speedup is approximately given by

$$S \approx \frac{T_I}{T_R + T_P/(k+1)}, \tag{3}$$

where S is the speedup, T_I is the Immediate Updating run time, and $T_L = T_R + T_P/(k+1)$ is the Lazy Updating run time. T_P approximates the time spent on updating propensities, T_R is the time required for the remaining work in a simulation (draw random numbers, store observations, etc.), and k is the effective number of skipped updates. Values of k are model dependent and pivotal for both, speedup and accuracy, as skipped updates often imply outdated propensities, implying less accurate amounts that can potentially snowball through the remainder of a simulation. Note that it is best to think of each of the parameters as expectations, because they all depend on the stream of (pseudo)random numbers used by SSAs. This was particularly obvious in the Birth Model, where the total number of actions (and hence computing time) depended on the random timing of the first few actions, which would either accelerate or delay overall population growth. Skipping updates in this context can modulate the total number of actions executed during a simulation.

Running an Immediate Update simulation gives us T_I, running a Lazy Updating simulation with a high tolerance approximately gives T_R (since $T_P/(k+1)$ presumably becomes very small). Naturally, the time spent updating propensities may be approximated by $T_P = T_I - T_R$. Eq. (3) does not directly describe how the speedup depends on the tolerance, but it does approximate how the speedup depends on k. Assuming the tolerance only affects the speedup through k, we can rewrite (3) as

$$S \approx \frac{T_I}{T_R + T_P/(k(\varepsilon_L)+1)}, \tag{4}$$

where we substitute $k(\varepsilon_L)$ for k, since k is a function of the tolerance ε_L. The function $k(\varepsilon_L)$ is unknown, so we decided to approximate it with a quadratic function

$$k(\varepsilon_L) \approx a{\varepsilon_L}^2 + b\varepsilon_L. \tag{5}$$

Substituting the quadratic approximation into the speedup equation yields

$$S \approx \frac{T_I}{T_R + T_P/(a\varepsilon_L^2 + b\varepsilon_L + 1)}. \tag{6}$$

Recall that we can estimate T_I, T_R, and T_P directly from observations, so that leaves just two unknowns: a and b. Measuring two Lazy Updating run times, T_{L1}, T_{L2} for two different intermediate tolerances, ε_{L1}, ε_{L2}, allows us to solve for a and b (equations not shown); inserting in Eq. (6) gives us $S(\varepsilon_L)$, the approximate speedup as a function of the Lazy Updating tolerance. Equation (6) was then tested against the different models. We obtained the best estimates when picking values of $\varepsilon_{L1}, \varepsilon_{L2}$, that led to very different speedups, but were not too close to either extreme (i.e., no speedup or maximal speedup). We tried a linear function, but found the quadratic function performed substantially better in many cases where we tested it in a defined range of values. The quadratic function in Eq. (5) assumes that speedups asymptotically approximate a maximum. This seems to work well for models A and C, but in the Birth Model we observed the opposite, which can easily lead to negative estimates of a. In that case the higher quality of predictions of the quadratic function applies only to a rather narrow range of ε_L; predictions outside of this range show large, unexpected fluctuations that are difficult to control. Using a linear function instead of Eq. (5) is generally less precise, but also less erratic and easier to control. It may thus provide a very useful first approximation of speedup that is particularly easy to obtain. Thus we apply this linear approach to predicting speedup in the Birth Model by setting a

$= 0$ in Eq. (5) and substituting

$$k\left(\varepsilon_L\right) \approx b\varepsilon_L \approx \frac{\varepsilon_L}{\varepsilon_{L1}}\left(\frac{T_P}{T_{L1}-T_R}-1\right) \tag{7}$$

in Eq. (4), where ε_L is the tolerance for which we wish to predict the speedup $S(\varepsilon_L)$ using the Lazy Updating computing time T_{L1}, which was observed for the intermediate tolerance ε_{L1}, and T_I, T_R, T_P are defined above. Thus, besides two extremes (T_I, T_L), the linear function requires only one additional observation T_{L1}, which is best chosen to represent an intermediate speedup. Together with the quadratic function that needs only two intermediate values, this shows that simple means are sufficient for obtaining useful rough estimates of speedups. Both approximations capture the general shape of the curve (assuming the quadratic function stays in its range). Nonlinear least squares methods result in even better predictions, but require more effort.

E. Accuracy measurements

Lazy Updating requires the user to specify tolerance ε_L, which determines how often propensity updates occur, and consequently, how accurate reported time series are. To test the accuracy of time series, we compared Lazy Updating to Immediate Updating for each model except for Model D. For all simulations, the Lazy Updating tolerance was set to $\varepsilon_L = 0.1\%$. We computed three sets of 16 000 runs for measuring accuracy, one set for Lazy Updating, and two for Immediate Updating. To evaluate the accuracy of Lazy Updating we calculated a p-value every 0.01 time units for testing the correctness of the (false) null-hypothesis that the measured amounts at that point for all 16 000 Lazy and Immediate Updating runs were sampled from the same distribution. We report a representative fraction of these results to avoid visually cluttering figures. We used the t-test to compare the means and the F-test to compare the variances between these two sets for each model[67] (both tests were two-sided, comparing equal numbers of simulations and leaving parameters at defaults provided by the R statistical programming language, version 3, http://www.r-project.org). We could have run more simulations to increase the sample size of our tests and thereby increase their statistical power until we would report a statistically significant difference. Given unlimited computing resources, it is easy to see that at some point even the minutest deterministic differences will become visible. We argue that such effort would be wasted, because it merely demonstrates what we already knew before the start: our null-hypothesis is wrong, since Lazy Updating and Immediate Updating are different simulation methods.

Most real-world simulation studies compare results from different parameter combinations that may lead to different outcomes of interest with respect to the questions that motivated the model. Using Lazy Updating in such a context would conflate uncertainty about effects of Lazy Updating with uncertainty about the effects of using a different parameter combination. Thus it is important to have a reasonably accurate overview of the potential consequences of using any approximate method like Lazy Updating. We limited our number of repeats to 16 000 and kept it constant in all our accuracy tests, to provide a very high repeat count for comparison with real-world simulation work. This use of constant sample sizes also facilitates comparisons of p-values across different tests in this study.[78]

To inform this tradeoff, modelers must know the size of uncertainties that Lazy Updating might generate. Towards this end we compared the means and standard deviations of amount differences in time series generated by Lazy Updating and Immediate Updating to help assess the size of differences and their importance in the context of a model. Highly significant differences may not matter for a question that motivated a model and differences that matter may not be significant (in a non-prohibitive number of repeat runs). To provide an additional point of reference we repeated all these analyses for two different sets of times series, independently generated by Immediate Updating (i.e., we know the null-hypothesis to be true; reported as light gray in figures to reduce visual clutter).

The effect of the Lazy Update tolerance on accuracy was also measured. We varied the tolerance and collected the data at the last time point of the simulations. The Lazy and Immediate Updating data were then compared for each of the tolerances as detailed above.

Additionally, we used Model C to test how the rate coefficient affects the accuracy of Lazy Updating in a contracting population. We used the t-test and F-test to look for differences in distributions at the time when the population was expected to reach half its initial value. The expected time was based on the ODE equivalent of Model C (see Fig. 10) and was determined by the rates used in the sets of simulations (0.01, 0.03, 0.1, 0.3, …, 300, 1000; each computed in the same three sets of 16 000 simulations as described above). Note that our simulations are stochastic and used Eq. (1) for computing propensities, resulting in a factor of 2 in our ODE equivalent from our use of the "n choose k" function in Eq. (1).

F. Simulation code

The Lazy Updating and Immediate Updating simulators were implemented in C++ using early prototypes of the Evolvix simulation infrastructure; the models were specified using a prototype of the Evolvix model description language (http://evolvix.org). The pseudorandom numbers were generated with the Mersenne Twister mt19937 32-bit[68] generator and the exponential and uniform random variate generators available from http://www.boost.org (boost version 1.54.0). The Lazy Updating tolerance was always $\varepsilon_L = 0.1\%$ (unless mentioned otherwise). For comparison we selected the following three algorithms: (i) Tau-Leaping,[42] as implemented in StochKit, version 2.0.10, http://stochkit.sourceforge.net;[69] (ii) the Optimized Direct Method (ODM),[33] also in StochKit; for Tau-Leaping and ODM, we let StochKit choose the best variants of each algorithm to use for the model; and (iii) the Sorting Partial Propensity Direct Method (SPDM) as implemented and originally made available by their authors (http://www.mosaic.ethz.ch/Downloads/pdm—now at http://mosaic.mpi-cbg.de/?q=downloads/stochastic_chemical_net).[30] I/O operations during speed measurements were not a substantial

factor in the run time, as we instructed all code to only record data at the last time point.

III. RESULTS

A. Large gains in speed for the ATP Model

Comparing speedups for different simulation methods: To get a sense of how each algorithm scales with the number of reactions and propensity updates, we compared their respective speedup for different chain-lengths of the ATP-Model in the context of the equivalent of many additional reactions on rare parts (Figs. 3(a) and 3(b); Model A_1). SDM with Lazy Updating and SPDM scale similarly, and they achieve a speedup of about 18 fold over SDM with Immediate Updating. For the most part Tau-Leaping has a similar speedup as ODM, but Tau-Leaping is faster around 1000 reactions. We believe this is due to the fast reactions not occurring as often, relative to the huge number of other reactions; this would allow Tau-Leaping to take larger steps. The same

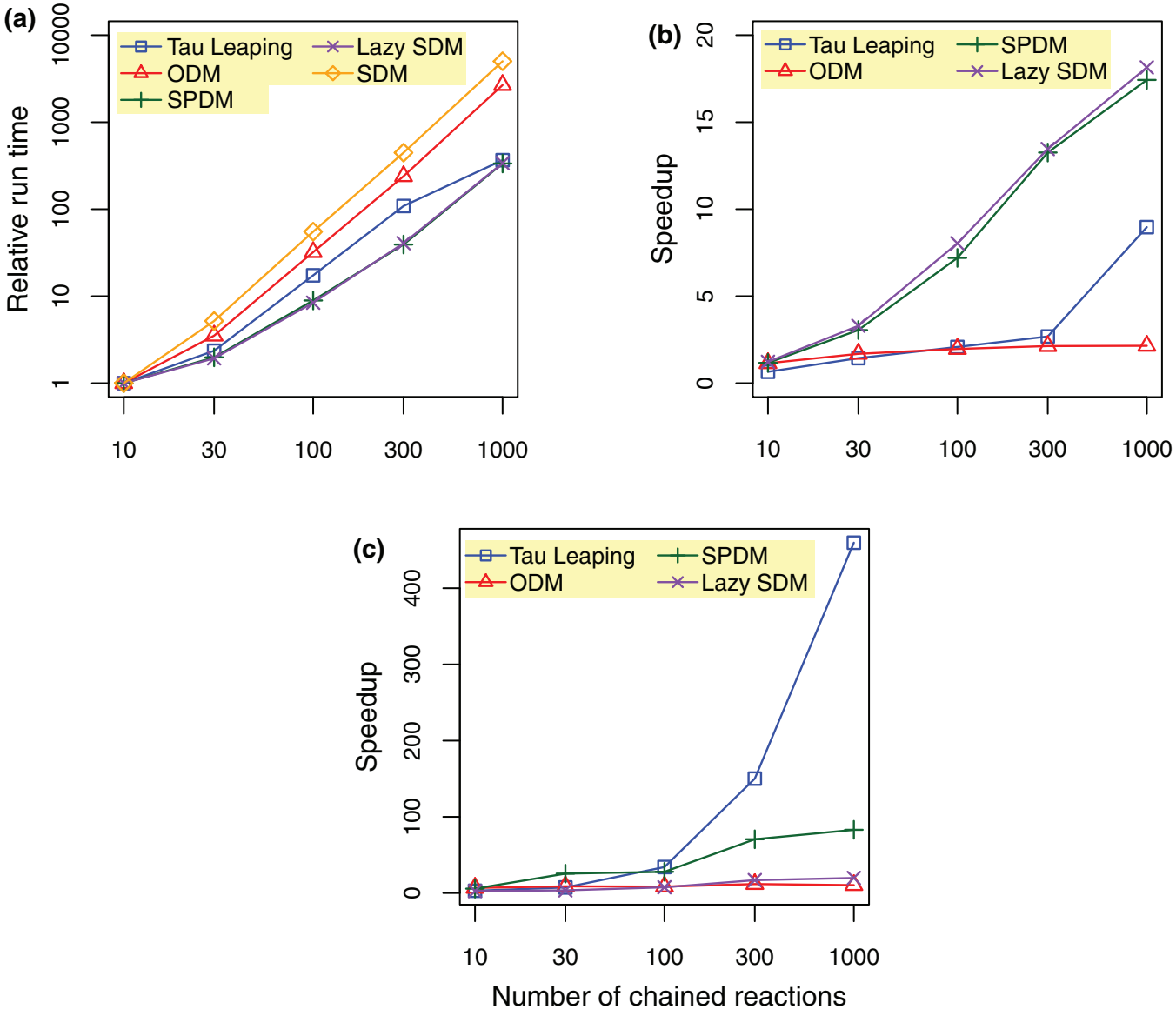

FIG. 3. **Comparison of the Sorting Direct Method with Lazy Updating to other stochastic simulation methods in the presence of hubs.** We use each ATP model with different hub sizes (connected reactions) to measure simulation time for the Sorting Direct Method (SDM; Immediate Updating), SDM with Lazy Updating (both our code), as well as the Optimized Direct Method (ODM), Tau-Leaping, and a Partial Propensity method (SPDM), all three as implemented by their authors. Each method has its own strengths and trade-offs. Partial Propensity is exact and fast, but tricky to implement and to our knowledge existing implementations cannot simulate third-order or higher reactions. While not exact, Tau-Leaping speed can exceed SPDM (see Model A_2)—if not slowed down by frequent propensity updates that involve rare parts (see Model A_1). Lazy Updating does not limit the order or reactions, is relatively easy to implement, and offers substantial speedups for certain types of models (Figs. 3(a) and 3(b); Model A_1; see Sec. IV). (a) **Comparisons of relative runtimes** for different sizes of **Model A_1**. The extremely fast reversible binding reaction in A_1 slows down Tau-Leaping by forcing small steps. SPDM and Lazy Updating deal with propensity updates more efficiently and therefore surpass SDM, ODM, and slowed down Tau-Leaping. We highlight how run times scale with model size by reporting run times relative to those required for the model with 10 reactions. (b) Different view of **Model A_1** data from above; all run times are relative to SDM *without* Lazy Updating. (c) Speedup for ATP **Model A_2 without frequent actions on rare parts.** Here Tau-Leaping steps are large enough for the larger systems, so Tau-Leaping takes a commanding lead in speed. See Fig. 2(a) for Models A_1 and A_2 and Methods for parameters.

model without the frequent reactions on rare parts shows an excellent Tau-Leaping performance, as Tau-Leaping steps can be much larger (Fig. 3(c)). At 1000 reactions, Tau-Leaping speedup is ≈460-fold over SDM with Immediate Updating. SDM with Lazy Updating performs similarly to the first version of Model A, and achieves ≈19-fold speedup. SPDM speedup is ≈83-fold.

Impact of Lazy Update tolerance on speed: To investigate the effect of the Lazy Update tolerance on run time, we compare SDM with Immediate Updating to Lazy Updating with multiple tolerances using Model A_1 with the fast reversible binding reaction and 300 reactions in the pathway (Fig. 4(a)). The maximum achievable speedup is about 25-fold. For the small tolerance of 0.01% the speedup is ≈24 fold, so large tolerances are not necessary to achieve a substantial speedup. The dotted line shows the expected speedup based on a few of the data points (see Table I and Sec. II), suggesting Eq. (6) captures the overall relationship between the tolerance and the speedup.

Impact of number of parts on speed: The amount of ATP has an effect on Lazy Updating's speedup (Fig. 4(b)), ranging from 8-fold for 10 000 ATP molecules to over 32-fold for 3×10^6 ATP molecules before leveling off.

Accuracy of Lazy Updating results: Time series generated by Lazy Updating for the ATP model are almost identical to those from Immediate Updating. In Fig. 5(a) we decided to compute our summary statistics only for 100 time series with Lazy Updating and 100 time series with Immediate Updating, as larger numbers would have made differences virtually disappear. We compared the means and variances of amounts at the end of simulations for three sets of 16 000 independent simulations using the *t*-test and *F*-test, respectively (Fig. 5(c); 1× Lazy Updating, 2× Immediate Updating). We also computed descriptive summary statistics for the same sets of simulations (Fig. 5(b)). In either case, differences between Lazy Updating and Immediate Updating were below the detection limit for Model A.

Conclusion: The speedups we see when using Lazy Updating for ATP in Model A do not seem to come with a substantial cost in accuracy for the parts impacted by the lazily updated part. This is not surprising, as we expect the model to be in quasi-steady state.

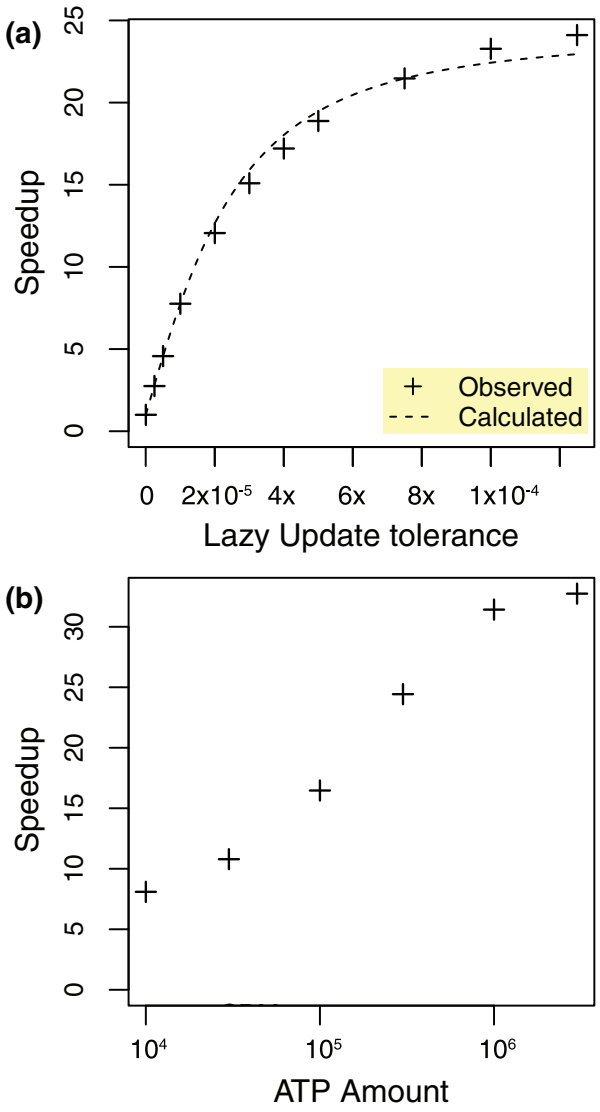

FIG. 4. **Dependency of speedup** on tolerance and hub molecule counts in the ATP Model, a case for which Lazy Updating was designed. (a) **Increasing speedup in the presence of a hub** for increasing Lazy Updating tolerances ε_L in Model A_1 with 300 reactions (see Fig. 2(a) and Sec. II for details). Crosses report means of 50 runs and the dashed line approximates expected speedup (see Eq. (6), Table I for calculation). (b) **Increasing hub amount improves speedup**. As the amount of ATP increases, SDM with Immediate Updating is slowed down much more than SDM with Lazy Updating ($\varepsilon_L = 0.1\%$, speedup is mean of 50 runs of A_1 with 30 reactions).

TABLE I. Estimated Speedup expected from Lazy Updating using Eqs. (4)–(7) with summary of parameters for the speedup predictions shown in Figs. 4(a), 6, and 8(a). Both, *S* and *k* depend on the Lazy Update tolerance (here $\varepsilon_L = 0.001$). Times were measured in seconds, averaging over 50, 5000, and 5000 simulations of the A_1 (300 reactions), B and C Model, respectively. Estimating parameters *a* and *b* requires two speedup measurements in addition to the frame established by measuring the extremes. For the Birth Model we used $a = 0$ to enable the easy conversion to a linear model that requires only one intermediate speedup measurement and is more robust, but less precise.[a]

Parameter	Meaning	ATP Model	Birth Model	Contraction
T_I	Total time under Immediate Updating only; $T_I = T_P + T_R$	42.38	0.183	0.0010
T_P	Time needed for propensity	40.62	0.154	0.00049
T_R	Time needed for rest	1.76	0.0288	0.00051
a	Use in $k \approx a{\varepsilon_L}^2 + b\varepsilon_L$ (Eq. (5))	2.30×10^{10}	0	2.83×10^6
b	Use in $k \approx a{\varepsilon_L}^2 + b\varepsilon_L$ (Eq. (5))	7.66×10^5	38.57	579.09
k	Average skipped updates at S	2.38×10^4	0.03857	3.41
S	Expected speedup at a given ε_L	24.09	1.03[a]	1.61

[a]For the Birth Model, the predicted *S* is in contrast to the 42% speedup observed at $\varepsilon_L = 0.001$; see also Fig. 6, and relevant sections on Birth Model in Secs. II and III.

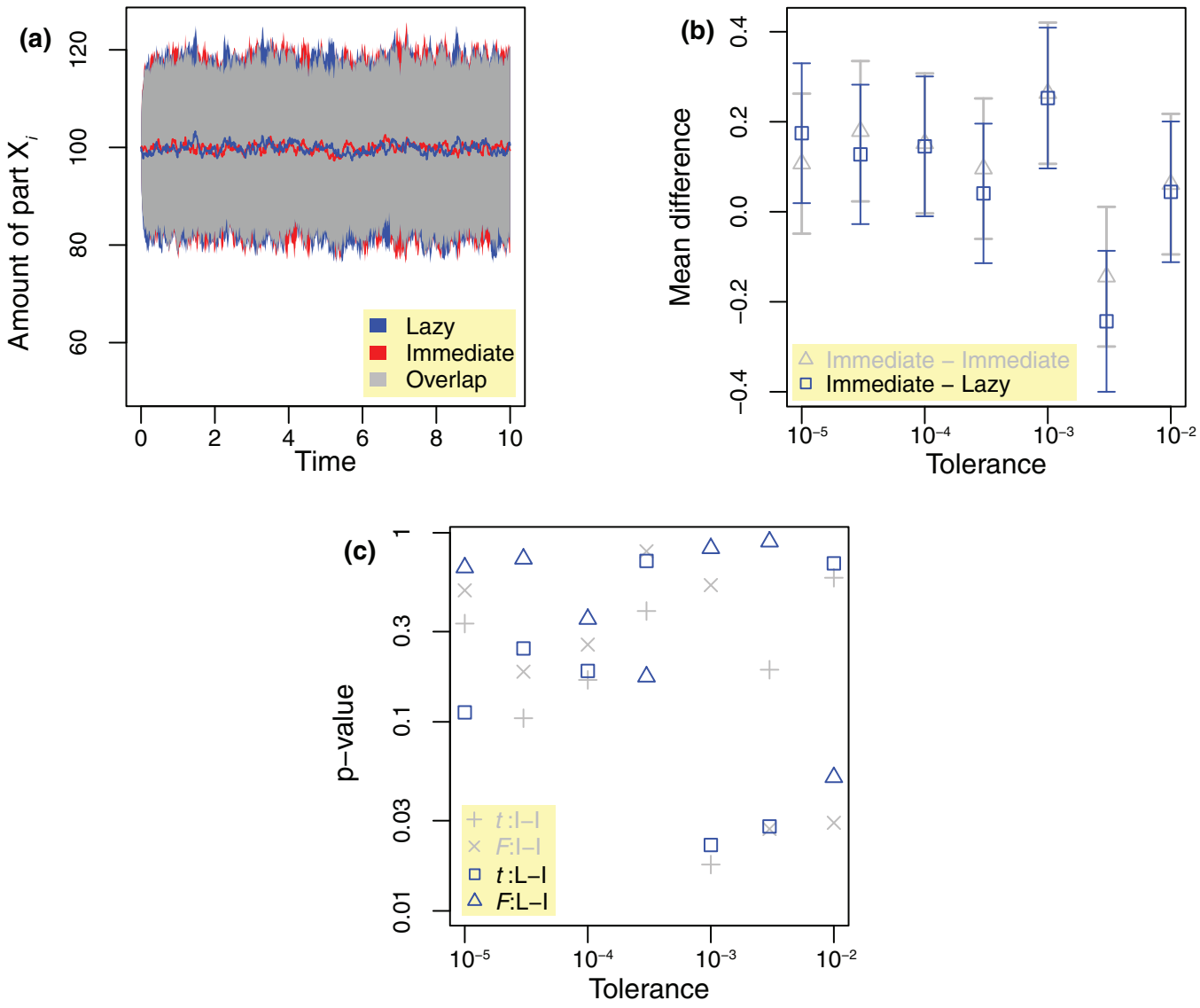

FIG. 5. **Accuracy of Lazy Updating for the ATP Model A_1** with 30 reactions. (a) **Time series of descriptive statistics** for Lazy Updating and Immediate Updating. Blue and red lines in the middle give means for 100 individual Lazy and Immediate Updating simulations, respectively; blue and red areas mark ±2 StDev, where gray area indicates overlap. Plotting only 100 runs gives a better sense of noise in the system than plots of $>10^4$ runs, where noise is almost invisible, even at large magnifications. Here we show amounts of X_7, a representative metabolite in the ATP Model, not the lazily updated part ATP itself ($\varepsilon_L = 0.1\%$). (b) **Absolute arithmetic mean differences** in amounts at the end of simulations between Lazy Updating and Immediate Updating for different tolerances (blue circles, line ±1 StDev) as compared for the three sets described below. (c) **Statistical tests** of the null-hypothesis that amounts of the part observed in Lazy and Immediate Updating simulations were both drawn from the same random distribution (blue squares and triangles indicate p-values for the *t*-tests and *F*-tests, respectively). We did not correct for multiple tests, so some moderately low p-values are expected among many comparisons. These simulations do not show statistically significant discrepancies from Lazy Updating at the p = 1% level. However, p-values decrease with increasing sample size when comparing two distributions with very small differences, so we theoretically expect *sufficiently large* computational efforts to reveal even the smallest differences between Lazy and Immediate Updating; whether they matter is another question. Panels (b) and (c) show three sets of runs ($n = 16\ 000$ each) of Model A_1 with 30 reactions, recording amounts at time 10; one set of Lazy Updating runs is complemented by two sets of independent Immediate Updating runs compared for providing a better intuition on the magnitude of internal stochastic noise in the system, see light gray triangles.

B. The Birth Model is a very simple "worst-case-scenario" for Lazy Updating accuracy

Speedup: Lazy Updating was not designed to speed up models like this, so we were surprised to see any speedup at all. The speedups for the Birth Model reported in Fig. 6 do not seem to level off as for the other models. Hence the Birth Model does not satisfy the assumptions underpinning Eqs. (3)–(7), making it harder to predict speedups. Observed speedups are modest (from 25% at $\varepsilon_L = 10^{-5}$ to 55% at $\varepsilon_L = 10^{-2}$) for small tolerances around our recommendation of 0.1%, and vanish as expected if the large amount of parts at the end of growth is still effectively in the Immediate Updating regime. Moderately larger speedups are possible, but require a blatant disregard for accuracy and can therefore not be recommended. This apparent lack of an asymptotic maximal speed in this model pushed the quadratic approximation (Eq. (6)) beyond its ability to predict speedup safely (carelessly chosen values from Figure 6 easily lead to erratic predictions; not shown). Therefore, we decided to use a simpler linear function for estimating speedup (Eq. (7)): while less accurate in detail, its overall behavior is much more

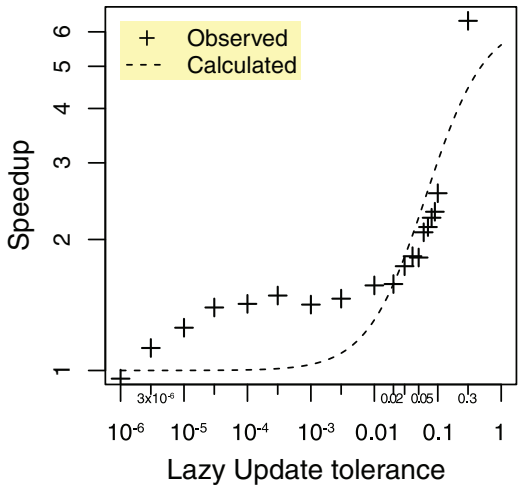

FIG. 6. **Speedup of Birth Model** for a given Lazy Update tolerance. A moderate speedup is expected for this model, when using Lazy Updating as recommended ($\varepsilon_L = 0.001$); the larger speedups reported here require reckless disregard of accuracy. Crosses report means of 5000 runs and the dashed line approximates expected speedup from the linear prediction (see Eqs. (4) and (7) and Table I for calculation of the figure, using $\varepsilon_L = 0.02$ as calibration point for computing k).

robust (but still depends strongly on the data point chosen to compute k).

Accuracy: For this model, Lazy Updating consistently underestimates the propensity. To assess the impact on accuracy, we used the same approach as detailed for the ATP Model and compared the means and variances of amounts for three sets of 16 000 independent runs using the t-test and F-test, respectively (Figs. 7(b) and 7(d); 1× Lazy Updating, 2× Immediate Updating). We also computed descriptive summary statistics for the same sets (Fig. 7(c)). Lazy Updating's accuracy is very high until about time 15 (Figs. 7(a) and 7(b)). This is due to '*de facto* Immediate Updates', which occur for populations $<1000 = 1/\varepsilon_L$ for $\varepsilon_L = 0.1\%$. Lazy Updating with that tolerance only starts at a population size of 1000 or more. As time passed, the absolute difference between amounts in time series from Lazy Updating and Immediate Updating grew consistently, as did the significance of the t-test. The p-values show one pattern until about time 15, which may be due to the end of "*de facto* Immediate Updates" in otherwise Lazy Updating simulations. For the Birth Model, the corresponding transition is made a bit before the systematic trends appear in Figure 7(b). After that, the p-value from the t-test between Lazy and Immediate Updating distributions started trending down towards more significance. Thus it is a question of time whether differences become significant in the Birth Model.

We compared only 100 simulations in Fig. 7(a); we also show 2 very thin lines from individual runs to highlight the low stochasticity at the late stages of growth. Given the large variance of these models, one might expect more rugged lines. However, the trajectories are quite smooth once the population size becomes large. Most of the variance stems from stochastic fluctuations early in the time series, when the stochastic noise is still large relative to the population size.

Impact of tolerance: We expected the tolerance to have a strong effect on the accuracy of the Lazy Updating simulations for the Birth Model, so we tested for significant differences at various tolerances, while recording some summary statistics about the difference between Lazy and Immediate Updating time series (Figs. 7(c) and 7(d)). Lazy Updating maintained a high level of accuracy until a tolerance of approximately 0.1%. Beyond that, it visibly diverged from Immediate Updating. When interpreting this, it is important to remember the existence of "*de facto* Immediate Updates." For $\varepsilon_L = 10^{-5}$, Lazy Updating reverts to Immediate Updating for population sizes below 10^5, a number that is only reached relatively late in our simulations that stop at time 30. The reversal of this statement suggests a powerful way of controlling the impact of Lazy Updating on stochastic simulations (see Sec. IV).

Conclusion: Lazy Updating leads to very small speedups for the Birth Model; its effects on accuracy depend on the context, but can easily become substantial, if growth continues for too long. We designed the Birth Model to reveal the limits of Lazy Updating accuracy, so these findings are no real surprise. Applying Lazy Updating to growing populations should be limited in time and must be carefully tested for potential losses of accuracy.

C. The Contraction Model shows systematic changes in accuracy

Like the birth model, the Contraction Model was made for testing the accuracy of Lazy Updating.

Speedups: Similar to the Birth Model, the speedup is modest (Fig. 8(a)); dissimilar to the Birth Model, Contraction Model speedups seem to have a more pronounced upper limit at around 2-fold. For our tolerance recommendation of 0.1% there is a speedup of around 1.6-fold. The initial amount also affects the speedup (Fig. 8(b)), but even a 100-fold increase in amount does not increase the speedup more than 2-fold in our model.

Accuracy: Again we compared 100 Lazy and 100 Immediate Updating simulations (Fig. 9(a)). We did not expect to see such smooth curves and such small standard deviations; after double-checking our code, we plotted two individual time series that are visible by zooming into Fig. 9(a)). In fact, the time series from this model do not have a large variance. As with the Birth model, we measured amounts in three sets of 16 000 simulations (1× Lazy Updating, 2× Immediate Updating). Fig. 9(b) reports the p-values from t-tests and F-tests over the course of the simulation; t-tests show temporarily increased differences between Lazy and Immediate Updating simulations. This is expected, considering that the Contraction Model starts at a fixed point with no initial variability, and after the distributions diverge, they once again come closer as the population approaches zero.

Impact of Tolerance: Plots of mean difference between Lazy and Immediate Updating simulations over tolerance show that Lazy Updating is virtually indistinguishable from Immediate Updating below a tolerance of about 0.01% (Fig. 9(c)). Lazy Updating consistently overestimates the propensity in this scenario, which is visible if the

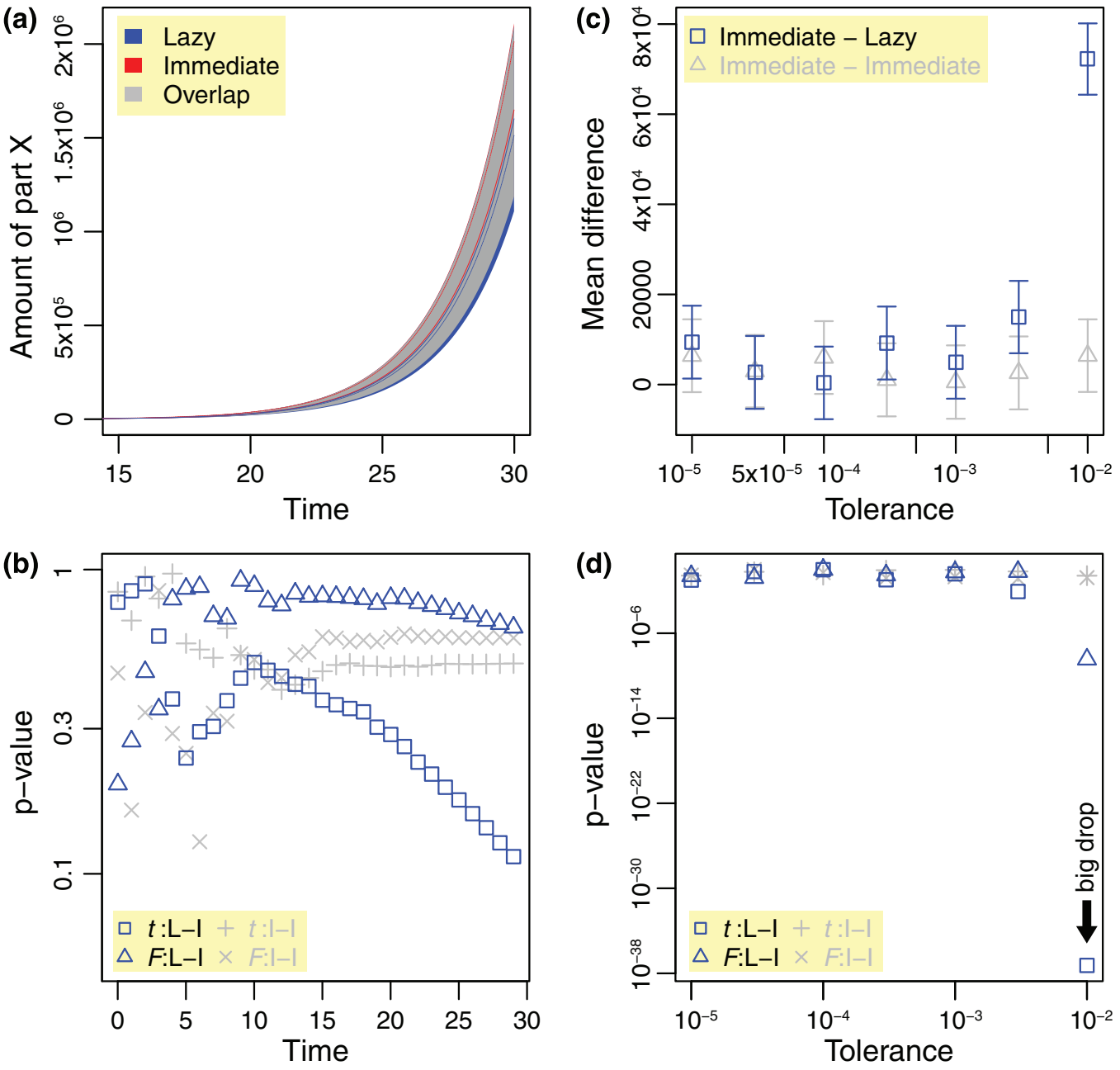

FIG. 7. **Accuracy of Lazy Updating for the Birth Model**, using analyses as in Fig. 5. (a) **Systematic delays from Lazy Updating grow over time**; to aid visibility, we only show data after time 15. Runs start at $X_0 = 10$ and use $\varepsilon_L = 0.1\%$, implying Lazy Updating reverts to Immediate Updating whenever $X < 1000$ ($= 1/\varepsilon_L$). Simulations starting at $X_0 = 1000$ did not yield substantially different results (not shown). As in Fig. 5(a), we average $n = 100$ runs; here we added two single-run lines (zoom in to see: Lazy, blue dashed-dotted line; Immediate, red dashed line) to highlight the low stochasticity during the later stages of these single runs. (b) **Statistical tests** of differences over time (not needed in Fig. 5) were inspired by the systematic bias visible as "delayed increase" in (a). We performed *t*-tests and *F*-tests along the time series for three sets of 16 000 simulations (1× Lazy Updating, 2× Immediate Updating) and while not significant for our results at $\varepsilon_L = 0.1\%$, there is a clear trend towards increasing significance in the *t*-tests, suggesting the need for checking the robustness of conclusions when using Lazy Updating in growing populations. (c) **Mean difference** between Immediate and Lazy Updating for different tolerances in the Birth Model after sampling amounts at time 30. Plots like this, help translate more abstract p-values into model related quantities. (d) **Test of the impact of tolerance** on the statistical power to detect differences in means and variances using *t*-tests and *F*-tests, respectively, on the data from (c).

contraction continues long enough. The p-values also become more significant as the tolerance increases (Fig. 9(d)).

Impact of contraction rate: To investigate the relationship between the contraction rate coefficient and the accuracy of Lazy Updating, we calculated p-values when the population was reduced by half (Fig. 10). As a point of reference, p-values between two sets of Immediate Updating time series were also calculated. These make it very clear that there are statistically highly significant Lazy Updating errors for this model, even though Lazy and Immediate Updating time series in Fig. 9(a) appear to be very similar.

D. Detailed metabolic model prototype and mechanisms of Lazy Update speedups

To test the real-world performance of Lazy Updating for complex models that might be encountered in biology,

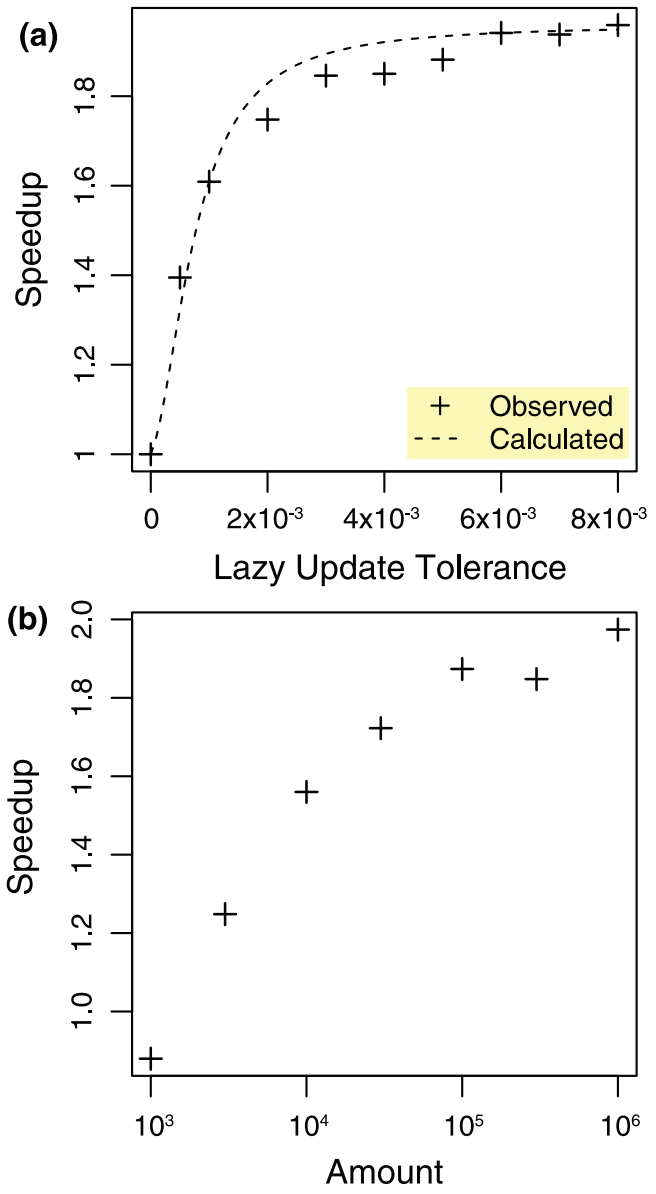

FIG. 8. **Contraction Model dependency of speedup** on (a) **tolerance** and (b) **initial amount** of part X. As Lazy Updating was *not* designed for models like this, only a small speedup is expected at the recommended maximum tolerance ($\varepsilon_L = 0.001$), which is ignored in (a) and used in (b). Both panels report means of 5 000 runs as crosses; the dashed line approximates expected speedup (see Eq. (6), Table I for calculation).

we applied Lazy Updating to a metabolic model prototype capturing aspects of human red blood cell metabolism.[65] A cursory look at the equilibrium concentrations and reaction rates of this model shows that both are spanning many orders of magnitude, a situation that likely reflects actual biology. Fig. 11 shows a visual impression of the size and connectivity of Model D. Fig. 12 presents the speedup for all three scenarios defined in Sec. II, as observed in three replicate simulations. We find that Lazy Updating of the few very moderate hubs in this model at a tolerance of 0.1% leads to a speedup of less than twofold. We hypothesized that maybe the presence of other very fast "non-hub" reactions might consume most of the CPU-time. To address this, we introduced Lazy Updating for the "1KFast" metabolites. We found that the speedup improved, but not as much as we had expected. When we added volume to the model, the picture did not change—much to our initial surprise. We think that this is caused by very frequent reactions that keep the CPU busy and leave little room for speedup by Lazy Updating. Indeed, simple calculations predicted that in equilibrium, many of the actions unaffected by volume had extraordinarily high propensities, confirming our interpretation. The picture changed substantially when we included temperature in the model: speedups of up to 20-fold were suddenly possible. Thus in principle detailed biological models and parameter combinations exist, where Lazy Updating can lead to substantial speedups that are likely to inspire work in new directions that would not have been pursued otherwise.

IV. DISCUSSION

Our results show that the Lazy Updating add-on for SSAs can lead to a 10-20× speedup of stochastic simulations of models with hubs (see Figs. 3, 4, and 12 and Sec. I, on hubs). Lazy Updating accomplishes this by avoiding costly propensity updates that contribute little to simulation accuracy. Lazy Updating speedups are small if hubs do not trigger frequent and costly propensity updates. For large speedups, models must meet all the following conditions:

(i) "obvious" hubs must exist in the structure of the network connectivity;
(ii) propensities affected by these hubs must be updated frequently;
(iii) these updates must consume a substantial fraction of the *total* simulation time when updated immediately.

Large speedups do not necessarily require large Lazy Update tolerances (ε_L) that substantially decrease accuracy. For some models, very small ε_L could get most of the speedup without sacrificing much accuracy (Figs. 4(a) and 12), but in others accuracy suffered substantially (Figs. 7 and 9). Thus, tradeoffs between speedup and accuracy are model-dependent and researchers will have to determine ε_L for each hub independently if they want to maximize speedup without sacrificing much accuracy.

Where Lazy Updating goes wrong. Based on the results reported here, users will find that Lazy Updating operates between the following extremes:

(i) Large speedups with a small loss of accuracy. This is expected for frequently accessed hubs that remain more or less at the same large amount. Lazy Updating was designed for parts in such a quasi-steady-state[70,71] (Figs. 4 and 5).
(ii) Small speedups and large loss of accuracy. This is expected when amounts change substantially as seen in the Birth and Contraction models (Figs. 6–9). If this behavior is combined with other complications like series of precise thresholds and timed switches, eventual errors can compound over time.

Here are some critical model properties that may help decide whether inaccuracies from Lazy Updating of

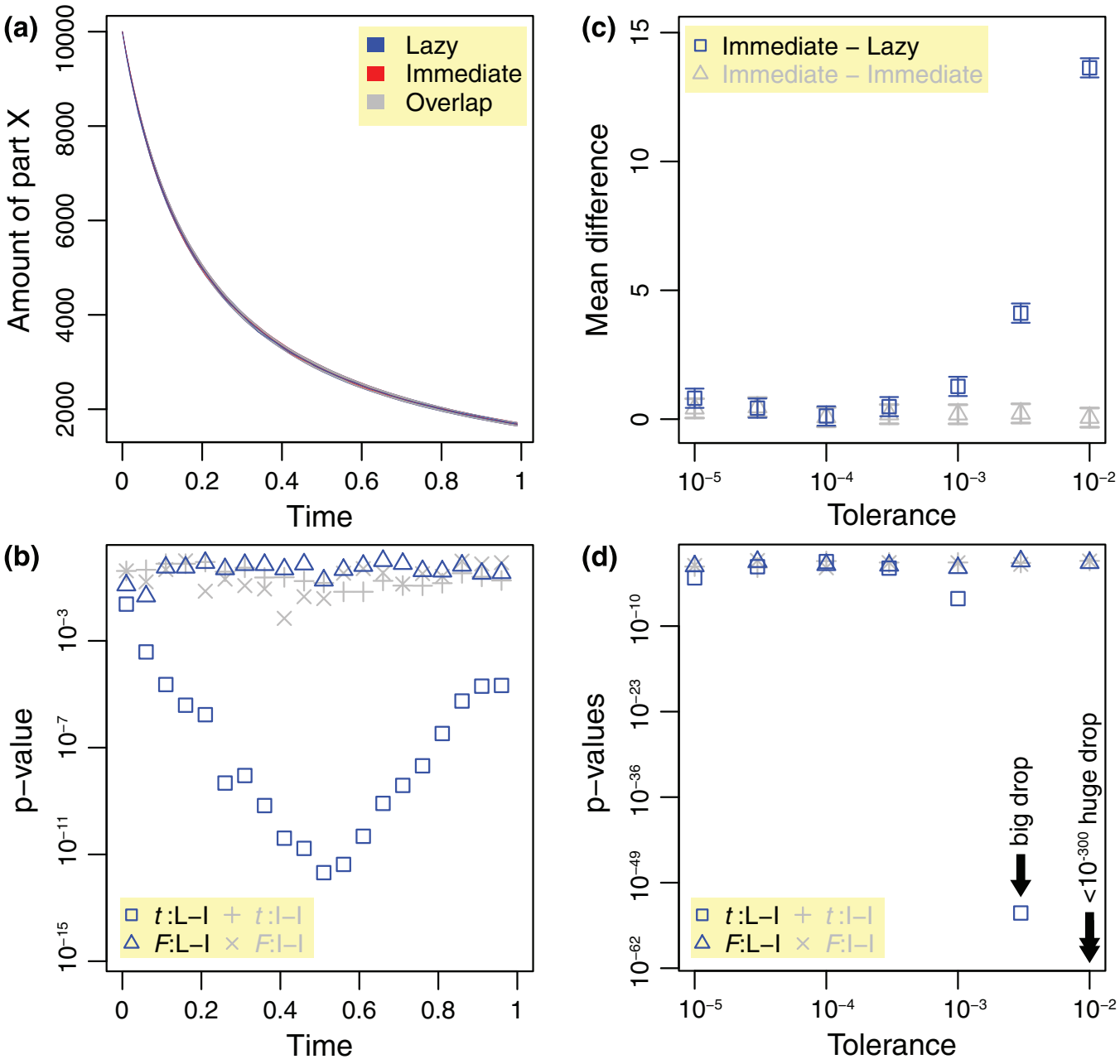

FIG. 9. **Accuracy of Lazy Updating for the Contraction Model** using analyses as in Fig. 5 and Fig. 7. (a) **Systematic delays**. As before, only 100 runs are shown to highlight the low variability at $\varepsilon_L = 0.001$ (zoom in to see two single-run lines: Lazy, blue dashed-dotted curve; Immediate, red dashed curve). (b) **P-values** of *t*-tests for different times indicate growing discrepancies of means between Lazy and Immediate Updating after all runs start identically ($X_0 =$ 10 000); after an intermediate maximum, discrepancies are compressed when approaching small amounts; *F*-tests indicate no discernable effects on variances ($n = 16\,000$; $\varepsilon_L = 0.1\%$). (c) **Mean difference** between Immediate and Lazy Updating for different tolerances in the Contraction Model after sampling amounts at time 1 ($n = 16\,000$). This plot highlights that the contraction model can lead to highly significant differences, even though their absolute size appears very small when compared to the Birth model, Figs. 7(c) and 7(d). (d) **Testing the impact** of ε_L on the statistical power to detect differences in means and variances using *t*-tests and *F*-tests, respectively, on the data from (c).

dynamically changing parts matter for a model. For example, low accuracy may be acceptable in dynamical parts during initial simulation burn-in if these parts reach the right amount before key results are recorded. In the Birth and Contraction models Lazy Updating reaches the same levels as exact SSAs, only a bit later (Figs. 7 and 9). Thus the appropriateness of Lazy Updating may hinge on the mechanisms that determine the new amount after a period of change for a lazily updated part P_{LU}. (see examples below). We identify the precise conditions that lead to large Lazy Updating errors as "**critical model properties for Lazy Updating**"; many more such properties are likely to exist, each with a special set of complications. Here we describe four examples of such critical properties based a combination of our results (Figs. 7 and 9) and modeling intuition:

(i) If the new amount of P_{LU} after a period of change is determined by system-inherent equilibriums and independent of the time available to reach it, then Lazy Updating will generate a slight delay in reaching the new amount,

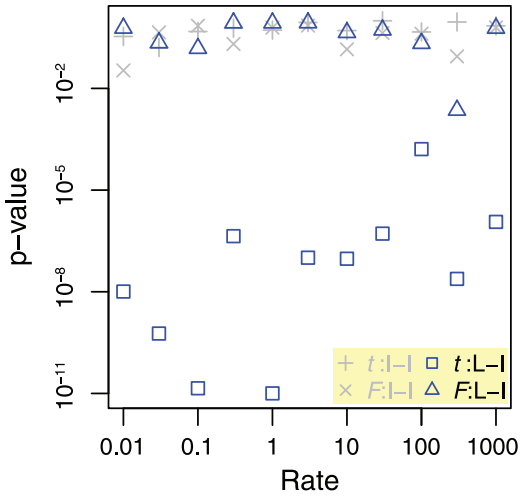

FIG. 10. **Effect of rate coefficients on Lazy Updating accuracy** in the Contraction Model (initial $X_0 = 10\,000$, $\varepsilon_L = 0.001$). Amounts for comparisons were recorded at time t when populations were expected to have declined to about X = 5000, as predicted from the equation $t = 2/(\text{Rate} * X_0)$, based on the ODE version of this model. We used Eq. (1) for computing the propensity in the stochastic model (the factor of 2 results from our use of the "n choose k" function in Eq. (1)).

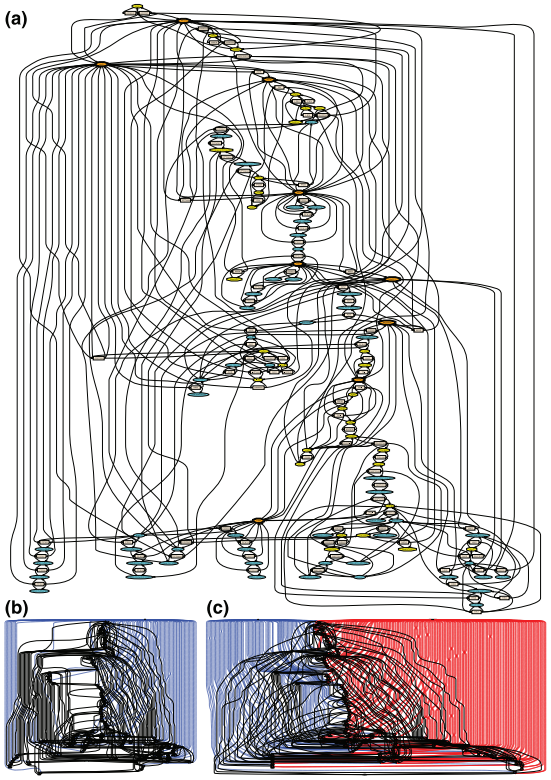

FIG. 11. **Detailed Metabolic Model Prototype** for testing Lazy Updating in more realistic settings. (a) **The published prototype model** with no modifications to its structure. (b) **Volume as a hub** (all blue lines) makes all non-first-order reactions dependent on the corresponding volume correction; all their propensities need to be updated each time one of the few import or export reactions occur that can increase or decrease volume, respectively. (c) **Temperature as a hub** (all red lines) in addition to volume. Here each action also depends on a relative temperature that is changed by each reaction, making temperature the most connected part in the system. We switched two groups of parts to Lazy Updating in all three prototypes: (i) "**5Hubs**" (zoom in: "orange triple-octagons") are involved in 5 or more reversible reactions that also include other parts. (ii) "**1KFast**" (zoom in: "yellow octagons") have equilibrium part amounts >1000 at the volume we used. Zoom in to see all other parts as "cyan ellipses" and actions in "white rectangles."

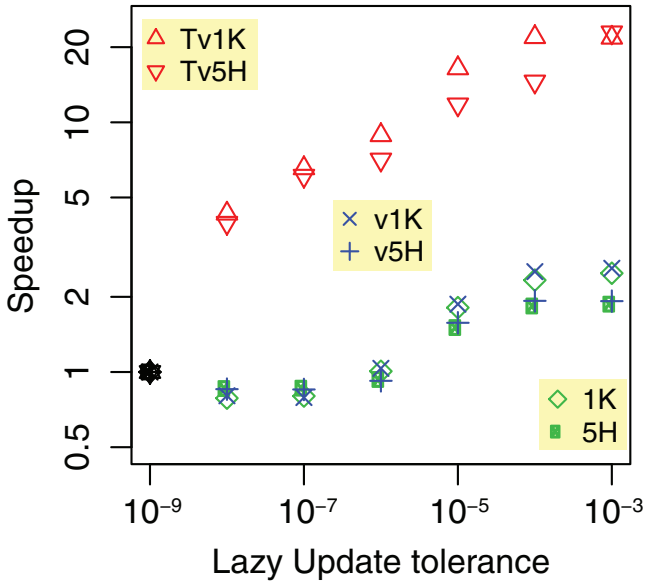

FIG. 12. **Lazy Update Speedup observed in the Detailed Metabolic Model Prototype**. (a) **Original networks** with Lazy Updating for "5Hubs" ("5H") and "1KFast" hubs ("1K") show speedups <2× (green symbols). (b) Corresponding networks with **volume dynamics** for both types of hubs ("v1K", "v5K", blue crosses) show almost no speedup over original (presumably actions without effects on volume drown out impacts of those affecting volume; this can be predicted from extrapolating the model's equilibrium propensities). (c) Corresponding networks with **temperature dynamics** ("Tv1K", "Tv5K", red triangles) showed effects up to 20×. All three networks (a–c) are shown in Fig. 11 and are here simulated each under Immediate Updating (for scaling), Lazy Updating for "5Hubs" and Lazy Updating for "1KFast" (see Fig. 11).

but does not affect the amount itself. The correct speed of changes is essential for the timing of switch-like processes that could be of direct modeling interest or that could indirectly impact the rest of the model.

(ii) If the new amount of P_{LU} after a period of change is determined by the time available to reach it, then the delays in changes caused by Lazy Updating will also affect the amount of P_{LU} after the change. This further increases chances that conclusions drawn from the model might be affected, as these inappropriately modified new levels can cause additional avalanches of twisted propensities.

(iii) If the critical conditions above are combined with triggers that require a certain amount to be present at a certain time (e.g., in the race of an immune response against an infection), then not meeting that strict requirement could essentially trigger anything in the model. In gene-regulation very large changes for a cell can essentially be switched on/off, and "unique small original" events in biology (e.g., infection with a single virus) can have potentially huge consequences (e.g., infection of a cell).

(iv) Additional examples include, but are not limited to competition between two growing strains of different bacteria.

In models with such critical properties, small differences can easily blow up into major effects.

Thus, we recommend switching off Lazy Updating for all parts that change with a recognizable speed that might have a remote chance of making a difference to the dynamics of interest in the model. If such a model still needs the speedup

from Lazy Updating, then its accuracy requirements have to be analyzed at a level beyond the scope of this study.

To choose ε_L *we recommend keeping relative Lazy Updating error tolerances below* $\varepsilon_L = 0.001$ *and apply it only to known hubs in quasi-steady-state* (i.e., such that none of the special cases discussed above apply[70,71]). All other parts should use Immediate Updating. Smaller ε_L are better and should be used where computationally feasible, unless evidence shows that reductions in accuracy do not affect modeling conclusions. Specifying the level of Lazy Updating by defining ε_L as a relative error tolerance, as done here, facilitates intuitive assessments of its impact on the accuracy of part amounts in stochastic simulations. As a consequence of Eq. (2) an update is triggered, every time the amount of part i changes when it satisfies

$$p_i < 1/\varepsilon_{Li}. \tag{8}$$

Thus our recommended maximum $\varepsilon_L = 0.1\%$ Lazy Update error tolerance implies that Lazy Updating is effectively switched off whenever a part has an amount of less than $1/\varepsilon_L = 1000$. The same effect can switch all parts to Immediate Updating by setting $\varepsilon_L = 0$, subject to the accuracy limits of the numerical types in Eq. (2).[79] Optimizing the tradeoff between accuracy and speedup beyond these simple guidelines requires specific comparisons of speedup merits and accuracy costs (see Secs. II and III, for examples).

Lazy Updating and other algorithms: It is easy to integrate Lazy Updating into other SSAs if data structures exist for looking up dependencies (e.g., as already stored in the Next Reaction Method;[18] see Sec. II for details). The fast Tau-Leaping methods compute *all* propensities for the time points they choose to jump to; they might be further accelerated by using Lazy Updating for models with hubs, which could be a potential area of future research.

Lazy Updating is an "**add-on**" and thus fundamentally different from other stand-alone simulation approaches such as Tau-Leaping methods,[24,42] Partial Propensity methods,[29–32] or more direct "Gillespie" methods.[11,20] Instead, Lazy Updating integrates into a "**base**" **method** and builds on that method's performance and accuracy by reducing the computations triggered by hubs. It does so by restructuring some simulation tasks in a way that is not easily compared to many other methods by treating different actions and parts non-uniformly.

In Tau-Leaping, leaps are **synchronous** in the sense that at the time a leap is jumping to, the execution of all actions and the calculation of all new propensities happen at the same time. This bundling of many actions renders changes of amounts to be approximate, as the number of times each type of action is expected to occur is computed with limited accuracy. In Lazy Updating, "leaps" are **asynchronous** in three ways: first, the changes in amounts caused by an action are applied immediately, whereas the corresponding updates of affected propensities may be calculated later; second, if an action changes amounts of different parts, then updates for the corresponding propensities of affected actions occur at different times, whenever needed or convenient; third, different actions are executed at the diverse times they occur at (i.e., asynchronously as with Immediate Updating), and are not bundled into groups for execution as in Tau-Leaping.

Thus Lazy Updating allows for **subtle errors in the timing** of actions that are otherwise **executed precisely** as under Immediate Updating. The causality chain for the timing error of *one* action from lazily updating *one* part may thus be described like this:

Cause: The update of the propensity for action A_k is delayed with these **effects**:

(1) The propensity for A_k used in the simulation is slightly too high or too low.
(2) The expected waiting time for the execution of A_k is too low or too high.
(3) The execution time of A_k is on average too early or too late.
(4) The amounts of parts changed by A_k occur on average too early or too late.
(5) The resulting propensity and timing errors ripple through the system.

If many such errors occur and do not cancel out on average, then Lazy Updating can delay larger trends (see the Birth and Contraction Models). If the precise timing of these trends might affect questions addressed by the model, then Lazy Updating should not be used (unless errors are demonstrated to be inconsequential).

Other ways of phrasing the differences are: Tau-Leaping can be activated "**per-action**" (as available parts and speeds of actions determine leap size), whereas Lazy Updating can be activated "**per-part**" (Lazy Updating of a hub has no impact on a precise simulation if the hub amount does not change; in addition it will track unforeseen changes in hub amount). Tau-Leaping holds part amounts and propensities constant for a specified time step, whereas Lazy Updating changes part amounts immediately and holds propensities fixed for different times (until amounts change enough to force propensity updates, or updates occur as side-effects of other actions). This is equivalent to stating that the **news that a hub amount has changed travels at varying speeds**, generating the slight timing errors discussed.

To illustrate these important insights into the mechanism of Lazy Updating, consider a single action updating X_1, X_2, and ATP in Model A. It will immediately update all *three* part amounts and the propensities of all actions that consume X_1 or X_2, but it will not update propensities for the many actions consuming ATP. Thus "news" concerning the new amount of ATP "travels slow" to these actions. However, news does not necessarily travel at the "lowest permissible speed" determined by ε_L, which forces updates of actions once ATP changes beyond the relative difference ε_L, thereby guaranteeing an upper local limit on accuracy errors from Lazy Updating. Indeed, this slowest permissible speed is often irrelevant. For example, if the next action updates X_3, X_4, and ATP, then propensities of actions consuming X_3 and X_4, are again updated immediately (and again do not include a general update of all actions that require ATP); however, this time updates of actions consuming X_3 and X_4 *already* incorporate previous news on ATP changes from previous actions (since propensities can only be computed in full and can only use

current amounts, which are kept up to date for hubs). While this substantially reduces errors for actions that occur often, Lazy Updating cannot avoid that propensities of actions that occur rarely enough will often be off by some factor determined by ε_L.

To summarize these mechanistic aspects of Lazy Updating for a single action:

(i) changes of all amounts are precise,
(ii) propensities of all actions depending on immediately updated parts are precise immediately after an update,
(iii) propensities of actions depending on parts lazily updated some time ago are slightly off by at most a factor affected by ε_L;
(iv) these slight errors in propensities will subtly skew times at which actions are expected to occur, implying that timings are expected to be slightly off, when executing the precise changes of amounts described in (i).

Controlling accuracy in Lazy Updating requires careful attention to these subtle timing errors, which only cancel out for parts in quasi-steady-state.[70,71] Note, that all error limits discussed are local, and no limits can be given for global errors of absolute amounts of affected parts, as the latter may compound many local errors in complex ways that may or may not effectively cancel out. This structure of errors is shared by ODE solvers, Lazy Updating, Tau-Leaping and other approximate methods; thus expertise in controlling such errors for other methods may also apply to Lazy Updating. Whether such errors matter is highly model dependent (see above).

Lazy Updating solves a different problem than Tau-Leaping by focusing on hubs. Tau-Leaping speeds past Lazy Updating if its leap condition allows for covering larger chunks of time in one step;[25,42] in that case it would skip many more propensity updates than Lazy Updating (Fig. 3). However, the impressive speed of Tau-Leaping critically depends on the absence of frequent actions that involve parts rare enough to force Tau-Leaping to use a slow exact SSA (see Figs. 3(b) and 3(c)). In its exact SSA mode, Tau-Leaping will update all hub-dependent propensities like any other Immediate Updating code (and could potentially receive help from Lazy Updating). Error tolerances of Tau-Leaping are not directly comparable to ε_L values for Lazy Updating due to the differences in synchronicity discussed above; thus we omitted direct comparisons. If Tau-Leaping cannot take large enough steps, a fast precise method might be preferable, as it does not "try to leap" (no pay off, if the attempts fail). Of the precise methods we tested, the partial propensity method SPDM seems to win due to its impressive speed (Fig. 3), but to our knowledge it has only been implemented for bimolecular reactions, making it impossible, to consume ATP in metabolic reactions catalyzed by enzymes (see http://mosaic.mpi-cbg.de/pSSALib/pSSAlib.html).[29–32] Thus **Lazy Updating appears to be the add-on of choice for simulations that need to be precise in general, *and* that have actions that require more than two parts, *and* that suffer from too many propensity calculations caused by hubs**. The lightweight nature of Lazy Updating makes its benefits easy to realize in a broad range of contexts.

Tau-Leaping, Lazy Updating, and SPDM are not alone in seeking to address the problem of increasing simulation speed for detail-rich models. Various multi-scale methods have been developed[72–74] for models containing processes that occur at very different temporal or spatial scales.[25,39,72–76] Since Lazy Updating is an "add-on," it is not a full method and much less a multi-scale method; however, it might be able to extend the capabilities of a multi-scale method such as the "Nested SSA"[77] if hubs cause excessive computation times. Future investigations incorporating Lazy Updating into various methods could use the opportunity to investigate new models that might benefit from Lazy Updating or might exhibit critical model properties that degrade accuracy in unexpected ways. From our results we anticipate that Lazy Updating will bring new classes of models within the reach of computation with reasonable accuracy.

Lazy Updating helps investigate important biological questions: As reviewed in the Sec. I, hubs such as ATP and other molecules, or volume and temperature can play prominent roles in biochemical systems. Changes in hubs can be seen as trends in commonly used resources or environmental conditions that affect and are affected by many small actions. Investigating trends in the amounts of hubs that are functionally linked to candidate fitness correlates[64] like energy consumption or survival, are of particular interest to evolutionary systems biology.[64] As shown in Sec. III, Lazy Updating makes such work much easier, by substantially reducing the computational cost from including frequently updated hubs. We anticipate easy access to Lazy Updating will encourage researchers to include more known hubs into increasingly realistic simulation models, ultimately leading to a deeper understanding of cellular energy metabolism, growth, survival, and evolution.

Conclusion: When used appropriately, Lazy Updating trades a small amount of accuracy for a substantial speedup. Implementing Lazy Updating within existing stochastic simulation algorithms is relatively simple. Lazy Updating can cut computational costs caused by frequently updated hubs that affect many actions. Thus Lazy Updating encourages the exploration of new biological questions that involve network hubs in models ranging from biochemistry to ecology and evolution.

ACKNOWLEDGMENTS

We thank David Anderson, Tom Kurtz, Bill Engels, James Faeder, Linda Petzold, two anonymous reviewers, and the other members of the Evolvix Development Team for stimulating discussions and helpful comments on aspects of this study. We particularly thank Seth Keel and Iratxo Flores-Lorca for their large contributions to the Evolvix code base without which this study could not have been completed. This work was supported by the National Science Foundation NSF CAREER (Award No. 1149123) to L.L. and by the Wisconsin Institute for Discovery at the University of Wisconsin-Madison. K.E. was also supported by the National Institute of General Medical Sciences of the National Institutes of Health under a Training Grant in Genetics, Award No. T32GM007133.

[1]E. Klipp, W. Liebermeister, C. Wierling, A. Kowald, H. Lehrach, and R. Herwig, *Systems Biology* (John Wiley and Sons, Hoboken, NJ, 2013).
[2]A. Cornish-Bowden, *Fundamentals of Enzyme Kinetics* (John Wiley and Sons, Hoboken, NJ, 2013).
[3]D. T. Gillespie, A. Hellander and L. R. Petzold, "Perspective: Stochastic algorithms for chemical kinetics," J. Chem. Phys. **138**, 170901 (2013).
[4]H. H. McAdams and A. Arkin, "It's a noisy business! Genetic regulation at the nanomolar scale," Trends Genet. **15**, 65–69 (1999).
[5]C. V. Rao, D. M. Wolf, and A. P. Arkin, "Control, exploitation and tolerance of intracellular noise," Nature **420**, 231–237 (2002).
[6]S. P. Otto and T. Day, *A Biologist's Guide to Mathematical Modeling in Ecology and Evolution* (Princeton University Press, Princeton, NJ, 2007).
[7]M. A. Nowak, *Evolutionary Dynamics: Exploring the Equations of Life* (Harvard University Press, Cambridge, MA, 2006).
[8]U. M. Ascher and L. R. Petzold, *Computer Methods for Ordinary Differential Equations and Differential Algebraic Equations* (SIAM, Philadelphia, PA, 1998).
[9]P. Moin, *Fundamentals of Engineering Numerical Analysis* (Cambridge University Press, Cambridge, UK, 2010).
[10]T. G. Kurtz, "The relationship between stochastic and deterministic models for chemical reactions," J. Chem. Phys. **57**, 2976–2978 (1972).
[11]D. T. Gillespie, "A general method for numerically simulating the stochastic time evolution of coupled chemical reactions," J. Comput. Phys. **22**, 403–434 (1976).
[12]D. T. Gillespie, "A rigorous derivation of the chemical master equation," Physica A **188**, 404–425 (1992).
[13]S. N. Ethier and T. G. Kurtz, *Markov Processes: Characterization and Convergence* (John Wiley and Sons, Hoboken, NJ, 2009).
[14]N. Van Kampen, *Stochastic Processes in Physics and Chemistry* (North-Holland, Amsterdam, NL, 2007).
[15]D. F. Anderson and T. G. Kurtz, "Continuous Time Markov Chain models for chemical reaction networks," in *Design and Analysis of Biomolecular Circuits*, edited by H. Koeppl, D. Densmore, G. Setti, and M. di Bernardo (Springer, New York City, 2011), pp. 3–42.
[16]D. J. Wilkinson, *Stochastic Modelling for Systems Biology*, 2nd ed. (CRC Press, Boca Raton, FL, 2012).
[17]D. T. Gillespie, *Markov Processes: An Introduction for Physical Scientists* (Academic Press, Waltham, MA, 1991).
[18]M. A. Gibson and J. Bruck, "Efficient exact stochastic simulation of chemical systems with many species and many channels," J. Phys. Chem. A **104**, 1876–1889 (2000).
[19]T. Lu, D. Volfson, L. Tsimring, and J. Hasty, "Cellular growth and division in the Gillespie algorithm," Syst. Biol. (Stevenage). **1**, 121–128 (2004).
[20]D. T. Gillespie, "Exact stochastic simulation of coupled chemical reactions," J. Phys. Chem. **81**, 2340–2361 (1977).
[21]D. T. Gillespie, "Stochastic simulation of chemical kinetics," Annu. Rev. Phys. Chem. **58**, 35–55 (2007).
[22]D. T. Gillespie, "Simulation methods in systems biology," Lect. Notes Comput. Sci. **5016**, 125–167 (2008).
[23]D. F. Anderson, "A modified next reaction method for simulating chemical systems with time dependent propensities and delays," J. Chem. Phys. **127**, 214107 (2007).
[24]D. T. Gillespie, "Approximate accelerated stochastic simulation of chemically reacting systems," J. Chem. Phys. **115**, 1716–1733 (2001).
[25]L. A. Harris and P. Clancy, "A 'partitioned leaping' approach for multiscale modeling of chemical reaction dynamics," J. Chem. Phys. **125**, 144107 (2006).
[26]S. Mauch and M. Stalzer, "Efficient formulations for exact stochastic simulation of chemical systems," IEEE/ACM Trans. Comput. Biol. Bioinf. **8**, 27–35 (2011).
[27]J. M. McCollum, G. D. Peterson, C. D. Cox, M. L. Simpson, and N. F. Samatova, "The sorting direct method for stochastic simulation of biochemical systems with varying reaction execution behavior," Comput. Biol. Chem. **30**, 39–49 (2006).
[28]A. Slepoy, A. P. Thompson, and S. J. Plimpton, "A constant-time kinetic Monte Carlo algorithm for simulation of large biochemical reaction networks," J. Chem. Phys. **128**, 205101 (2008).
[29]R. Ramaswamy and I. F. Sbalzarini, "A partial-propensity variant of the composition-rejection stochastic simulation algorithm for chemical reaction networks," J. Chem. Phys. **132**, 044102 (2010).
[30]R. Ramaswamy, N. González-Segredo, and I. F. Sbalzarini, "A new class of highly efficient exact stochastic simulation algorithms for chemical reaction networks," J. Chem. Phys. **130**, 244104 (2009).
[31]R. Ramaswamy and I. F. Sbalzarini, "A partial-propensity formulation of the stochastic simulation algorithm for chemical reaction networks with delays," J. Chem. Phys. **134**, 014106 (2011).
[32]R. Ramaswamy and I. F. Sbalzarini, "Fast exact stochastic simulation algorithms using partial propensities," AIP Conf. Proc. **1281**, 1338–1341 (2010).
[33]Y. Cao, H. Li, and L. Petzold, "Efficient formulation of the stochastic simulation algorithm for chemically reacting systems," J. Chem. Phys. **121**, 4059–4067 (2004).
[34]M. W. Sneddon, J. R. Faeder, and T. Emonet, "Efficient modeling, simulation and coarse-graining of biological complexity with NFsim," Nat. Methods **8**, 177–183 (2010).
[35]H. Li, Y. Cao, L. R. Petzold, and D. T. Gillespie, "Algorithms and software for stochastic simulation of biochemical reacting systems," Biotechnol. Prog. **24**, 56–61 (2008).
[36]M. Rathinam, L. R. Petzold, Y. Cao, and D. T. Gillespie, "Stiffness in stochastic chemically reacting systems: The implicit tau-leaping method," J. Chem. Phys. **119**, 12784 (2003).
[37]D. F. Anderson, "Incorporating postleap checks in tau-leaping," J. Chem. Phys. **128**, 054103 (2008).
[38]Y. Cao, D. T. Gillespie, and L. R. Petzold, "The slow-scale stochastic simulation algorithm," J. Chem. Phys. **122**, 014116 (2004).
[39]J. S. Hogg, L. A. Harris, L. J. Stover, N. S. Nair, and J. R. Faeder, "Exact hybrid particle/population simulation of rule-based models of biochemical systems," PLoS Comput. Biol. **10**, e1003544 (2014).
[40]J. R. Faeder, M. L. Blinov, and W. S. Hlavacek, "Rule-based modeling of biochemical systems with BioNetGen," Methods Mol. Biol. **500**, 113–167 (2009).
[41]Y. Cao, D. T. Gillespie and L. R. Petzold, "Adaptive explicit-implicit tau-leaping method with automatic tau selection," J. Chem. Phys. **126**, 224101 (2007).
[42]Y. Cao, D. T. Gillespie, and L. R. Petzold, "Efficient step size selection for the tau-leaping simulation method," J. Chem. Phys. **124**, 044109 (2006).
[43]L. A. Harris, A. M. Piccirilli, E. R. Majusiak, and P. Clancy, "Quantifying stochastic effects in biochemical reaction networks using partitioned leaping," Phys. Rev. E **79**, 051906 (2009).
[44]B. Alberts *et al.*, *Molecular Biology of the Cell* (Garland Science, New York City, 2007).
[45]S. Agarwal, C. M. Deane, M. A. Porter, and N. S. Jones, "Revisiting date and party hubs: Novel approaches to role assignment in protein interaction networks," PLoS Comput. Biol. **6**, e1000817 (2010).
[46]J. R. Karr *et al.*, "A whole-cell computational model predicts phenotype from genotype," Cell **150**, 389–401 (2012).
[47]A. I. Goranov *et al.*, "The rate of cell growth is governed by cell cycle stage," Genes Dev. **23**, 1408–1422 (2009).
[48]W. F. Marshall *et al.*, "What determines cell size?," BMC Biol. **10**, 101–122 (2012).
[49]T. R. Kiehl, D. Shen, S. F. Khattak, Z. Jian Li, and S. T. Sharfstein, "Observations of cell size dynamics under osmotic stress," Cytometry Part A **79**, 560–569 (2011).
[50]S. Hohmann, M. Krantz, and B. Nordlander, "Yeast osmoregulation," Methods Enzymol. **428**, 29–45 (2007).
[51]F. Montefusco, O. E. Akman, O. S. Soyer, and D. G. Bates, "Modeling and analysis of feedback control mechanisms underlying osmoregulation in yeast," in *A Systems Theoretic Approach to Systems and Synthetic Biology II: Analysis and Design of Cellular Systems*, edited by V. V. Kulkarni, G. B. Stan, and R. Raman (Springer, New York City, 2014), pp. 83–116.
[52]J. M. Mitchison, "Growth during the cell cycle," Int. Rev. Cytol. **226**, 165–258 (2003).
[53]L. Sherwood, H. Klandorf, and P. Yancey, *Animal Physiology: From Genes to Organisms*, 2nd ed. (Brooks/Cole, Belmont, CA, 2013).
[54]M. P. Gillum, D. M. Erion, and G. I. Shulman, "Sirtuin-1 regulation of mammalian metabolism," Trends Mol. Med. **17**, 8–13 (2011).
[55]C. Cantó and J. Auwerx, "Targeting sirtuin 1 to improve metabolism: All you need is NAD^+?," Pharmacol. Rev. **64**, 166–187 (2012).
[56]W. Wang, H. Wang, Y. Zu, X. Li, and T. Koike, "Characteristics of the temperature coefficient, Q_{10}, for the respiration of non-photosynthetic organs and soils of forest ecosystems," Frontiers Forestry China **1**, 125–135 (2006).
[57]T. Kätterer, M. Reichstein, O. Andrén, and A. Lomander, "Temperature dependence of organic matter decomposition: A critical review using literature data analyzed with different models," Biol. Fertil. Soils **27**, 258–262 (1998).

[58]E. D. Buhr, S.-H. Yoo, and J. S. Takahashi, "Temperature as a universal resetting cue for mammalian circadian oscillators," Science **330**, 379–385 (2010).
[59]G. Irvine, E. R. Lamont, and B. Antizar-Ladislao, "Energy from waste: Reuse of compost heat as a source of renewable energy," Int. J. Chem. Eng. **2010**, 627930.
[60]R. Kothari, V. V. Tyagi, and A. Pathak, "Waste-to-energy: A way from renewable energy sources to sustainable development," Renew. Sustain. Energy Rev. **14**, 3164–3170 (2010).
[61]J. S. Lim, Z. Abdul Manan, S. R. Wan Alwi, and H. Hashim, "A review on utilisation of biomass from rice industry as a source of renewable energy," Renew. Sustain. Energy Rev. **16**, 3084–3094 (2012).
[62]L. Loewe and W. G. Hill, "The population genetics of mutations: Good, bad and indifferent," Philos. Trans. R. Soc. B Biol. Sci. **365**, 1153–1167 (2010).
[63]L. Loewe, "How evolutionary systems biology will help understand adaptive landscapes and distributions of mutational effects," in *Evolutionary Systems Biology*, edited by O. S. Soyer (Springer, New York City, 2012), pp. 399–410.
[64]L. Loewe, "A framework for evolutionary systems biology," BMC Syst. Biol. **3**, 27 (2009).
[65]N. Jamshidi and B. Ø. Palsson, "Mass action stoichiometric simulation models: Incorporating kinetics and regulation into stoichiometric models," Biophys. J. **98**, 175–185 (2010).
[66]J. D. Orth, I. Thiele and B. Ø. Palsson, "What is flux balance analysis?," Nat. Biotechnol. **28**, 245–248 (2010).
[67]D. Wackerly, W. Mendenhall, and R. Scheaffer, *Mathematical Statistics with Applications* 7th ed. (Thompson Brooks/Cole, Belmont, CA, 2008).
[68]M. Matsumoto and T. Nishimura, "Mersenne Twister: A 623-dimensionally equidistributed uniform pseudo-random number generator," ACM Trans. Model. Comput. Simul. **8**, 3–30 (1998).
[69]K. R. Sanft *et al.*, "StochKit2: Software for discrete stochastic simulation of biochemical systems with events," Bioinformatics **27**, 2457–2458 (2011).
[70]C. V. Rao and A. P. Arkin, "Stochastic chemical kinetics and the quasi-steady-state assumption: Application to the Gillespie algorithm," J. Chem. Phys. **118**, 4999–5010 (2003).
[71]L. A. Segel and M. Slemrod, "The quasi-steady-state assumption: A case study in perturbation," SIAM Rev. **31**, 446–477 (1989).
[72]W. E, *Principles of Multiscale Modeling* (Cambridge University Press, Cambridge, UK, 2011).
[73]Y. Cao, D. Gillespie, and L. Petzold, "Multiscale stochastic simulation algorithm with stochastic partial equilibrium assumption for chemically reacting systems," J. Comput. Phys. **206**, 395–411 (2005).
[74]E. L. Haseltine and J. B. Rawlings, "Approximate simulation of coupled fast and slow reactions for stochastic chemical kinetics," J. Chem. Phys. **117**, 6959–6969 (2002).
[75]J. Li and M. Kwauk, "Exploring complex systems in chemical engineering—the multi-scale methodology," Chem. Eng. Sci. **58**, 521–535 (2003).
[76]J. O. Dada and P. Mendes, "Multi-scale modelling and simulation in systems biology," Integr. Biol. **3**, 86–96 (2011).
[77]W. E, D. Liu, and E. Vanden-Eijnden, "Nested stochastic simulation algorithm for chemical kinetic systems with disparate rates," J. Chem. Phys. **123**, 194107 (2005).
[78]It is beyond the scope of this paper to anticipate the enormous diversity of scheduling strategies used for real simulation work that is always forced to find a workable compromise between these two limiting extremes: know (almost) everything about the stochasticity of a single parameter combination or know nothing about the stochasticity of every parameter combination that can be simulated. Thus we just use $n = 16\,000$ for our tests of simulation stochasticity.
[79]A true $\varepsilon_L = 0$ is only possible for exact SSAs using arbitrary precision numbers. Any "exact" SSA using 64 bit floating point numbers effectively will introduce errors that would dwarf any accuracy errors caused by Lazy Updating with $\varepsilon_L \approx 10^{-15}$ when used in simulations with part amounts beyond $\approx 10^{15}$.